AUGMENTING GRAPHIC DESIGN PRACTICES FOR EXPRESSIVE VISUALIZATION AUTHORING

A Dissertation
Presented to
The Academic Faculty

By

John Thompson

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing
Department of Computer Science

Georgia Institute of Technology

DECEMBER 2020

AUGMENTING GRAPHIC DESIGN PRACTICES FOR EXPRESSIVE VISUALIZATION AUTHORING

Thesis committee:

Dr. John Stasko, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Zhicheng Liu
Department of Computer Science
University of Maryland

Dr. Alex Endert
School of Interactive Computing
Georgia Institute of Technology

Dr. Alberto Cairo
School of Communication
University of Miami

Dr. Keith Edwards
School of Interactive Computing
Georgia Institute of Technology

Date approved: November 10, 2020

For my brother Scott

ACKNOWLEDGMENTS

First, I would like to thank my wife, Lucy, for all of her support, patience, and understanding during this process. Lucy would always listen to my musings about new ideas, or my frustrations over a programming bug. Most importantly, Lucy helped me during those trying moments when I doubted myself, by reminding me about my passion for this work. I wholeheartedly share this achievement with Lucy, I could not have done this without her.

Thank you to my parents, Don and Cecilia, who gave me the opportunity to choose my own path in life and instilled in me the values required to achieve my goals. My Dad, who I attribute my perfectionism to; and my Mom, who always supported me in following my passions.

Of course, I am thankful for the guidance and friendship of my advisor, John Stasko. John has taught me so much about information visualization, human computer interaction, and how to be successful as an academic researcher. Over the years, John guided my work by urging me to think of the larger impact, instead of getting lost in the details.

I also owe a generous thank you to Zhicheng "Leo" Liu. Not only did Leo serve on my committee, but he was also my mentor during two summer internships with Adobe Research and the duration of my doctoral candidacy. In many ways, Leo acted as my second advisor. From Leo, I learned what was required to produce impactful research – Leo has demonstrated dedication and hard work, and how to mold research ideas into meaningful contributions.

I want to also acknowledge the guidance, instruction, and feedback that I received from my committee members. Alex Endert taught me the fundamentals of information visualization during multiple years of coursework. I also attribute much of the direct manipulation concepts implemented in these systems to Keith Edward's course on user interface design. Finally, I want to thank Alberto Cairo for encouraging me to aspire toward a real-world impact for this research.

Thank you to my collaborators at Adobe: Mira Dontcheva and Wil Li for their guidance on both projects; Alan Wilson, Sam Grigg, James Delorey, and Bernard Kerr for sharing their ideas and support on Data Illustrator. I would like to thank my co-authors on the critical reflections paper: Donghao Ren, Bonghsin Lee, Matthew Brehmer, Jeffrey Heer, and Arvind Satyanarayan who share my passion for visualization authoring tools.

As a graduate student, I also had the opportunity to grow as a designer and developer through jobs with world-class organizations. During my NASA Jet Propulsion Laboratory internship, I had the good fortune to collaborate with the superbly talented Alex Sciuto and Pei Liew, and to be advised by Scott Davidoff, Maggie Hendrie, Santiago Lobeyda, and Hillary Mushkin. Also during my fellowship with the CDC, I gained experience applying visualization communication concepts for public health under the guidance of Steven Sumner.

I also want to acknowledge Bobby Cohanim, Laura Major, Emily Vincent, Trevor Savage, and Jana Schwartz from C.S. Draper Laboratory who sparked my interest in human computer interaction.

Thank you to my lab-mates at Georgia Tech for their support, feedback on papers and talks, and most of all their friendship. Each of my lab-mates contributed to my dissertation in many different and important ways.

Finally, thanks are due to the following funding sources that made this work possible: the National Science Foundation via award IIS-1320537 and academic gifts from Adobe Research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Data visualization provides an effective method to tell stories, simplify complex concepts, support arguments, and make boring facts exciting. As communicative data visualization matures as a field in the computer graphics era, it has broadened into adopting practices from related fields such as graphic design. Designers employ a combination of methods to create expressive data visualizations in static or animated forms. These methods, however, are mismatched with their creative practice (e.g. writing textual code), or lack in generative power (e.g. graphic design tools). This dissertation explores opportunities to design and implement data visualization authoring tools for graphics designers. The goal of this research is to broaden practice and participation in data visualization. This dissertation contributes two interactive systems, Data Illustrator and Data Animator, that provide graphic designers with an understandable, feasible, and effective method to author expressive data visualizations (without writing textual code). These systems augment interfaces and interactions from graphic design practices to provide control over expressive frameworks for static and animated data graphics. I demonstrate each system's expressivity with a variety of example data visualizations and their usability through re-creation studies with designers. Finally, I reflect on the assumptions, strengths, and limitations of our approach, and identify future research directions for communicative data visualization.

CHAPTER 1
INTRODUCTION

People have used data visualization as an effective method to communicate data far before the proliferation of computer-generated graphics. Data visualization is effective for presenting data as it amplifies cognition by leveraging our visual perception channel. As communicative data visualization matures as a field in the computer graphics era, it has broadened into adopting practices from related fields such as graphic design, user experience design, art, cinematography, graphics animation, and journalism [1]. This diverse group of visualization designers have introduced novel forms of media, exposition techniques, and pushed the creative bounds of data visualization [2]. These new forms of data visualization bring exciting challenges for researchers to support the authoring of novel, expressive data visualizations for communication purposes. The opportunity to design and implement authoring tools to expand the bounds of this creative space is the crux of this dissertation. I seek to empower graphics designers to create expressive visualizations for communication without needing to write textual code. In pursuit of this goal, I design and implement user interfaces and interactions for authoring explanatory visualizations.

Explanatory visualization has permeated into the public eye through periodicals, blogs, business presentations, marketing material, and popular culture [3]. This exposure has lead to a renewed effort to understand and support explanatory visualization in the research community [4, 5, 6, 7]. Historically, visualization research has focused on *exploratory* visualization for analysis purposes. Used during analysis, visualization helps people to identify trends, correlations, find outliers, and make sense of data.

Exploratory visualization is often a precursor to communicating data. Analysis helps to identify key insights that an author will later present to their audience. Explanatory visualization can be thought of as an aid or lens that visualization authors use to transfer the

knowledge they have distilled during their data exploration process. The visualizations used in analysis do not necessarily transfer over to the communication stage. At times, authors throw away visualizations used during analysis, or build upon those visualizations for their ultimate design. Visualization designers do not necessarily follow a linear process from exploratory to explanatory visualization. More commonly, designers employ an iterative process moving between analysis and design of their final communicative composition [8]. This dissertation deals with designing and implementing authoring tools for explanatory visualizations.

Explanatory visualizations occupy a wide variety of forms used in a diverse set of practices [9]. For example, a bar chart can vary in its level of sophistication: from an unaltered template chart to a graphic with multiple layers, annotations, and embellishments. The number of visualizations can also vary from a single chart to a set of charts in a data narrative. Designers might also choose to use static graphics or include interaction and animation. I use the term *expressivity* to refer to the set of possible data visualizations that a tool can feasibly realize, regardless of the user. *Visualization authoring* refers to the process of constructing a visualization from data, with a design already determined by the designer. Visualization authoring tools allow designers to construct data visualizations. These tools range from interactive systems to programming languages and toolkits, or even plain old pen and paper. When considering which authoring tool to use designers might ask: (1) "Can the tool support the authoring of my visualization design?" and (2) "How easy is the tool to learn and use?" The first question corresponds to the expressivity of the tool, expressivity measures the lexicon of possible data visualizations that a tool can realize, regardless of the user. The latter question relates to how understandable and effective the tool is for target users, taking into account their experience and technical skills.

In this dissertation, I address a portion of these visualization forms, sets of practices, creation tasks, and user groups. My work focuses on supporting people who practice graphic design yet lack experience in programming computer-generated graphics. In this

specific set of practices, I will focus on communicating data via customized static visualizations and animated transitions between visualizations. By scoping this problem, I will assess if our target users, graphic designers, find the proposed authoring tools to be an *understandable* and *effective* method for creating *expressive* static and animated data visualizations. Importantly, this scoped version of the problem does not discount additional audiences from finding these authoring tools to be understandable and effective.

Currently, graphic designers employ a number of authoring tools to create explanatory visualizations. These tools allow designers to author expressive data visualizations that communicate their dataset's insights. Furthermore, some of these tools allow designers to tell data narratives, with animated transitions between story points. However, these tools are often mismatched with designers' experience, their creative practice, or lack the generative power required for complex visualizations. Here I discuss tools for authoring *static* visualizations, then follow up with a discussion of the tools that also support *animation*.

Authoring Static Visualizations – Designers often adopt programming toolkits or declarative languages [10, 11, 12] that accommodate the generative power and flexibility needed to create expressive static visualizations. These tools require time and resources to learn. Furthermore, programming requires authors to go back and forth between text and graphic representations of their design. Designers prefer to work directly with the visual representation of graphics instead of a programmatic way of thinking [8, 13]. Tools are needed that support the generative power of visualization toolkits such as D^3, yet do not require programming.

Interactive visualization authoring systems have begun to fill this gap. *Template-based* systems allow for rapid authoring of data visualizations. A number of online systems (e.g. Flourish [14], DataWrapper [15], Plot.ly [16], Raw [17]) support relevant sets of visualizations catered to their audience. While these tools provide quick specification of visualizations, they limit graphic designers to a set of available visualization types and afforded customizations. *Shelf construction* systems such as Tableau [18] allow authors to map data

fields to visual channels (e.g. position, color, shape). The system generates a valid chart given the author's specification using a grammar or algebra. These grammars have a data first pipeline: they start with data operators and subsequently visual marks appear in a later stage of the pipeline. Graphic designers approach visual design from the graphics first, mixing in data at later stages to encode data attributes to graphic elements. This dissertation adopts a graphics-first approach for authoring expressive data visualizations, aligning with graphic designers preferred workflow.

Graphic designers often resort to tools designed for their practice: vector editing tools and drawing tools to author static visualizations (e.g., Adobe Illustrator [19], Sketch [20]). For example, professional vector editors enable designers to work with shape geometries at the granular level of anchor points and curve segments. These tools also support designers to structure the placement of visual elements using grid systems, smart guides, and groupings. An effective authoring tool should leverage the rich set of concepts and tools from the graphic design community, while introducing new generative data binding concepts that fit into the creative tradition. These tools should feel familiar to designers to improve learnability and overall user experience.

A new generation of visualization authoring systems has sought to empower designers to create expressive, static visualizations in a *visual builder* environment [21, 22, 23, 24, 25, 26, 27]. Similar to my approach, these systems provide a visual interface to creating static, expressive data visualizations without writing textual code.

I contribute the Data Illustrator system for authoring static visualizations. Data Illustrator augments familiar vector editing tools with lazy-data bindings to ensure flexible yet powerful generation of data graphics. In Section 2.2, I will elaborate on how Data Illustrator compares and contrasts to other authoring systems and toolkits. Data Illustrator is a unique stake in the ground within the visual builder space.

Authoring Animated Visualizations – The available authoring systems that support animated data visualizations are few and far between. Current professional solutions for

creating animated data graphics requires programming knowledge or time intensive hand-encoding of data-driven animations.

The D^3 toolkit has quickly risen in popularity for creating interactive and animated data visualizations [28]. This success is in part due to the use of the "enter, update, exit" paradigm of D^3 that provides a composable and flexible method for declaring how objects enter, update, or exit from a visualization. Despite the expressive possibilities for this approach, it still requires programming knowledge. Furthermore, authoring animations by writing textual code requires constant recompiling and then replaying of the animation to compare design alternatives. In my work, I introduce how control over playback features can close this feedback loop for specifying and evaluating animations.

Graphic designers resort to tools designed for their practice such as visual effects and motion graphic tools (e.g., Adobe After Effects [29]) or prototyping tools (e.g., Adobe Xd [30], InVision [31]) to create animated visualizations. Visual effects tools allow designers to create animations by specifying shapes' visual properties at points in time as keyframes – intermediate frames are tweened by the system. However, these graphic animation tools have limited support for data binding. Manual-specification of animated visualizations is time intensive. An effective authoring system should leverage the rich set of concepts from the graphic design community, while introducing new generative data binding concepts that fit into the creative tradition. Again, these tools should feel familiar to designers to improve learnability and overall user experience.

Few interactive authoring systems exist that support authoring data-driven animations [32, 33, 14]. These systems are limited by the scope of visualizations and animations that one can construct. Before my approach, there has yet to be a visual builder tool that supports animation design.

Data Animator is the first visual builder tool for authoring animated visualizations. In Data Animator, animated transitions are specified between static data graphics created in Data Illustrator. The system automatically matches objects based on the relationships be-

tween objects across two visualizations. By default, matched objects animate by tweening visual properties and applying animation effects (e.g., fade in, fade out) to the unmatched entering or exiting objects. This approach supports rapid generation of animations through automation. To afford the authoring of data-driven temporal pacing, Data Animator supports staggering by data where the value of a data attribute determines the delay of the animating objects. I also introduce a concept called hierarchical keyframes, where the allocated duration for a transition cascades as a linear function of the parents' duration. Hierarchical keyframes allow the creation of expressive pacing by combining staging, staggering and grouping of graphical objects. The output medium of Data Animator is an animated data story that readers can step through via interactive controls.

Together, these two systems, Data Illustrator and Data Animator, represent my approach for investigating the feasibility of authoring tools for graphic designers in creating static and animated data visualizations for communication purposes.

1.1 Thesis Statement

I propose that authoring systems can be designed that are *understandable* and *effective* methods for authoring expressive data visualizations for communication purposes. In this dissertation, I will explain my approach for designing and implementing these authoring systems. The following four steps guide my approach:

1. Survey the design space to learn from exemplar data graphic compositions.

2. Understand designers' authoring approach for these example data graphics.

3. Design a cohesive framework that adheres to graphic design practices.

4. Augment user interfaces and interactions from design familiar tools.

The underlying frameworks for each tool must be understandable to graphic designers and effectively support the expressive set of data visualizations that target users wish

to create. To achieve this goal, each framework borrows from familiar design tool concepts, yet incorporates novel data-driven concepts that maintain congruence with the practice of graphic designers. I derive each new concept by conducting formative research on the authoring process of example data visualizations. Furthermore, these new data-driven concepts should be composable building blocks that designers can *understand* and use in tandem to create sophisticated designs.

My approach requires that each framework concept has an appropriate interface and interaction design. Framework concepts that extend expressive capabilities, yet do not fit within graphic designers' practice are discarded for more *understandable* framework concepts. My hypothesis is that by seamlessly introducing data-driven concepts within familiar design tools, graphic designers will be able to *effectively* author visualizations.

With this approach I will demonstrate the feasibility of two systems, Data Illustrator and Data Animator, that graphic designers are able to understand and effectively author expressive static and animated data visualizations.

1.2 Research Questions

To structure this dissertation, I introduce three research questions that guided my research direction. In the subsequent sections we provide evidence on how my approach answers these questions.

Research Question 1 (RQ1) – What are the building blocks to a static and animated visualization framework that graphic designers can understand and employ?

Defining a visualization framework that graphic designers can understand and employ requires a formative process to understand the intricacies of expressive visualizations that graphic designers currently create and how they would go about creating them in an authoring tool. As a result of this formative work, I propose two associated visualization frameworks for authoring static and animated data visualizations. The framework for ani-

mated visualizations extends the underlying static framework.

I argue that we have defined a visualization framework that graphic designers can understand and employ to create expressive static and animated visualizations. Visualization frameworks define the bounds of possible designs by describing the ways in which data can be transformed and rendered as a static and animated graphics. Frameworks intrinsically influence the manner in which authors create visualizations. The goal is to define visualization frameworks that extend expressive capabilities yet fit into the practice of graphics designers.

My proposed framework for static visualizations, the Data Illustrator framework, follows a graphics-first orientation where data is only incorporated into graphics as necessary. Furthermore, this framework allows for flexible specification of graphical properties outside of data-driven encodings by supporting lazy data bindings. My approach augments vector editing tools to select graphic primitives in the scene graph and bind data. The framework supports the definition of structured hierarchy of data through composable operators that bind and repeat/partition graphical shape by data. These data binding operations can be applied for a list of data rows, cloning each shape per row in the data table or binding a nested data structure to each repeated shape.

The proposed framework for animated visualizations builds upon the static framework from Data Illustrator. Data Animator supports the specification of animated transitions between two static visualizations created in Data Illustrator. Objects' between the two visualizations are automatically matched using an algorithm that analyzes the graphical and data relationships of objects. The product of the matching algorithm are matched objects that animate based on tweening the differences between visualizations, and unmatched objects that enter or exit with preset animation effects (e.g., fade in, fade out). The Data Animator framework relies on keyframe animation to coordinate the timing of each objects' transition. The framework introduces a concept called hierarchical keyframing that leverages the relationships of objects such as collections, groups, and sets of peer shapes

to create expressive pacing by combining staging, staggering and grouping of graphical objects.

Research Question 2 (RQ2) – Leveraging this framework, is it feasible to design and develop authoring systems that support designers to author expressive visualizations (without writing textual code) and fit within the tradition of graphic designers?

To improve the visualization authoring process of graphic designers, I propose two authoring systems embedded in the practice of graphic design. The first authoring tool, Data Illustrator supports authoring of expressive static visualizations. The second system, Data Animator supports the incorporation of static visualizations from Data Illustrator into animated visualizations. Together Data Illustrator and Data Animator can realize an expressive array of visualizations in static or animated presentation form. Furthermore, we designed these tools to be embedded within the practice of graphic designers. While designing these systems, we consider the ways in which we can directly borrow from this tradition, and how we must imagine new user interface and user interactions that appeal to graphic designers. Using insights from two series of design studies for each system we created prototypes, mockups, and discussed how to feasibly design these authoring systems.

To design each authoring system, we consider the following design goals: *familiarity, interpretability, discoverability, and control*. We believe these design goals are crucial for learnability, usability, and buy-in from graphic designers. *Familiarity* ensures that the user's previous experience will match their expectations; therefore if the tool uses a feature from an existing design tool, we want that feature to be consistent in appearance and behavior. In the case when a novel feature is needed, the design should be interpretable by the user. *Interpretability* requires the result of a user action to be immediately comprehended, an important requirement during generative data binding or data-driven animation. *Discoverability* on the other hand ensures that the interface design shows affordances for interaction so that users can detect the possibility of an action with a visual object. Finally,

users need to feel that they are in *control* at every step of the process to realize their design, especially when the system automatically generates aspects of their work.

Realizing these design goals requires the balance of tensions between opposing system concepts. For example, generative programming concepts are needed to repeat, bind, or animate graphics by underlying data, however automatic generation of graphical states is at odds with the unrestricted and direct illustration tasks of graphic designers. Another point of tension is ensuring understandability and ease-of-use while still supporting an expanding horizon of expressive visualization designs. Expanding system features and multiplying interfaces can result in an inconsistent, incohesive, and confusing system. Care is needed to include composable generative operations in the systems' interfaces and interactions. Finally, generative concepts enforce a programmatic or top-down approach to creating visualizations, this is in stark contrast to the bottom-up approach that graphic designers prefer. We reconcile this tension throughout the design process for both systems.

Research Question 3 (RQ3) – Can graphic designers understand and effectively use these tools to author expressive visualizations?

I evaluate my approach to support graphic designers in authoring expressive static and animated visualizations. Evaluating authoring systems is different from traditional systems as they provide open-ended authoring for uncounted design possibilities. Consequently, evaluation techniques with participants are often limited to a small subset designs to account for reproducibility and time constraints. Ren et al. reflect on evaluation techniques they employed for visualization authoring tools [34].

My research goals aim to evaluate the expressivity and usability (how understandable and effective) of these tools. From these methods I chose to evaluate the usability of my approach with re-creation studies with graphic designers. In particular, I evaluate Data Illustrator and Data Animator based on the participant's ability to understand and use the framework to author visualizations. The ability to think and act in terms of each framework

is the cornerstone of authoring with each tool. I chose to evaluate the expressivity through example galleries and video demonstrations for Data Illustrator and Data Animator. In the related works chapter I will elaborate on the trade-offs and decisions that led to the selection of these evaluation techniques.

1.3 Overview

In Chapter 1, I have discussed the research motivation behind visualization authoring tools for graphic designers. I also proposed my thesis statement with three research questions that guided my approach for designing understandable and effective static and animated visualization authoring tools. In Chapter 2, I discuss the current space of communicative visualization and highlight relevant research and commercial approaches that support visualization authoring for static and animated forms. I will also conceptually compare and contrast the proposed systems to relevant work in depth. Next in Chapter 3, I will present the Data Illustrator system, including the lazy data binding framework, the scope of the tool, design criteria and goals, the resulting interface and interaction design, details about the software behind the system, and finally the evaluation results of the system. In Chapter 4, I present the Data Animator system, including the object matching algorithm for quickly generating animated transitions between two static visualizations, a novel interface for viewing and disambiguating the results of the matching algorithm, and a timeline interface that supports the combination of multiple pacing techniques (e.g., staging, staggering, varying speed) when coordinating the motion of many data-driven objects. Finally in Chapter 5, I conclude by reflecting on my approach, discussing the merits and limitations of this research work, and introducing directions for future research on communicative visualization.

1.4 Attributions

1. The content of Chapter 3 is the result of a collaboration with Zhicheng Liu, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr and John Stasko. © 2018 ACM. Reprinted from *Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '18)* [35], Apr. 2018.

2. The content of Chapter 4 is partially the result of a collaboration with Zhicheng Liu, Wilmot Li and John Stasko. © 2020 Computer Graphics Forum, the Eurographics Association and John Wiley & Sons Ltd. Reprinted from *Understanding the Design Space and Authoring Paradigms for Animated Data Graphics, Computer Graphics Forum (EuroVis '20)* [36], July 2020.

3. The content of Chapter 4 is partially the result of a collaboration with Zhicheng Liu and John Stasko. Reprinted from *Data Animator: Authoring Expressive Animated Data Graphics, Under review at the ACM Conference on Human Factors in Computing Systems (CHI '21)* [37], Apr. 2021.

CHAPTER 2

RELATED WORK

This work extends upon previous research in visualization authoring tools, programming toolkits, declarative grammars, and data storytelling. Furthermore, we augment authoring paradigms, user interfaces, and user interactions from interactive tools from graphic design practices. Below, I highlight related work from research and industry starting with a general discussion of the state of the art in data visualization for communication. I then give an overview of tools with data-driven support for visualization construction (e.g, programming toolkits, declarative grammars, interactive authoring systems). I differentiate the tools in this discussion into two sections: first, I consider the strengths and weaknesses of these tools for authoring static visualizations; second, I highlight which of these tools support animation authoring. In this survey of related work, I note the approaches we build upon and differentiate our work from previous research and commercial products. Finally, I discuss graphic design practices in relation to data visualization, highlighting interactive design tools (without data support) that influenced our approach. In that final section, I compare the user interfaces and interactions from graphic design practices that our approach augments in Data Illustrator and Data Animator.

2.1 Data Visualization for Communication

2.1.1 Expressive Data Graphics

In this dissertation, I group unique visualizations and infographics into the category of *expressive data graphics*. As data visualization becomes further intertwined with the field of graphic design and information graphics, the two forms inextricably overlap. According to Chan, infographics explain complicated processes or concepts, while visualizations

generally represent the trends of larger datasets [38]. Chan claims that infographics are subjective while data visualizations are objective. Using a different approach, Taiei distinguishes infographics along the declarative & conceptual axis versus visualizations along the exploratory & data-driven axis [39]. I contend both axes represent the expressive data graphic space. Clear cut lines do not exist; there are different ends of the overlapping spectrum. In my research, I look at works that span the expressive data graphic space – with a focus towards data visualizations.

Due to the recent proliferation of visualizations and infographics, the research community has taken an interest in analyzing and understanding the effects of expressive data graphics. For example, Borkin et al. analyzed what makes a visualization memorable [40, 41], while Byrne et al. identified acquired codes of meaning that leverage the observer's previous experiences [42]. Skau et al. analyzed the effects of bar chart embellishments on the communication of the charts' data [43]. The existing research indicates that aspects of expressive data graphics are effective forms of communicating data.

2.1.2 Data Storytelling

As visualization becomes a cross-disciplinary field, traditions and methods from other disciplines have emerged. Data storytelling introduces narratives stories into visualization products to enable new user experiences around data. Researchers have recently catalogued data stories from online journalists, advocacy groups, and graphic designers. Segel and Heer systematically reviewed the design space of data stories to identify technique-based genres such as interactivity and messaging [4]. Stolper et al. conducted a similar survey to identify how authors tell stories by the following methods: communicating narrative, explaining data, linking separated story elements, enhancing structure and navigation, and providing controlled exploration [6]. Lee et al. take a step back to identify the design process for visual storytelling [5].

From these surveys, we learn that data stories come in many different forms. Giorgia

Lupi describes a notion of multilayered, non-linear storytelling through "info-spatial" compositions that she frequently employs in her work [44]. On the other hand, data journalists employ narrative controls (e.g., scrollytelling, steppers) that couple interactive animated visualizations with textual annotations in an article. In addition, animated infographics are used in presentations and branding to help reveal complex ideas as digestible bits of information. Storytellers also find ways to invite users to explore the data with the "martini glass method" that opens a linearly-directed story into an explorable interface [4]. Other authors restrict the interactions of the user to playing and pausing a data video.

Once seen as a beneficial way to engage users, data journalists have recently thrown cold water on interactivity [45]. Gregor Aisch recently commented at the Information+ conference that only "10-15% of users interact with stories" [46]. Aisch's editor at *the New York Times*, Archie Tse, also commented that "if you make a tooltip or rollover, assume no one will ever see it" [47]. Aisch has since clarified that "interactive graphics are still great," claiming that interaction builds trust in the data and allows users to gain further details [48]. Interaction can enhance a data story, but due to its infrequent use authors should primarily focus on revealing visual insights to users in a passive manner afforded by more narrative forms of interaction (e.g., scrollytelling, data videos, steppers).

Researchers have recently analyzed the narrative structure of data stories to identify effective story paradigms. Hullman et al. report on the beneficial effects of framing for data stories [49], and later identify sequential dynamics of narrative visualizations [50]. McKenna et al. investigated how user control of narrative flow influences the user's retention of the data story and experience [51]. Control flow interaction differs from explorable interaction, as the user controls the pacing of the data story with varying levels of granularity. Moritz Stefaner claims that telling a story does not automatically imply a simplistic, author-driven, linear, narration as we should consider ways to "[weave] data presentations into traditional storytelling formats, and bring rhetoric, dramaturgy, and suspense to data visualization" [52]. People want to tell more than one insight and create a narrative that

ties together visualizations that capture the observer's attention and leads them through data discovery.

2.1.3 Effects of Animation

The proliferation of animation in data narratives hints to animation's benefits for communicating data. It seems a reasonable assumption that if a static visualization can help viewers understand data then a moving visualization should be even better [53]. But what specifically about animation helps to benefit data graphics? Where is the line drawn between animation that clarifies and enriches versus that which obfuscates and distracts? We seek to support designers in creating rich, compelling, and clear animations rather than animating for animation's sake. To that end, I summarize the current body of knowledge on the advantages and disadvantages of animation for data graphics.

Recent research empirically explores pacing techniques for visualization. Previous findings from Heer and Robertson demonstrate the efficacy of staging animations to help viewers track objects and estimate changing values [54]. Fisher's synopsis provides further support for staging as a logical approach to assist viewers in understanding the intermediate steps of complex transitions [53]. More recently staging has been shown to improve subjects' accuracy to identify and disambiguate transitions that aggregate data [55].

In contrast to positive benefits for staging, Chevalier et al. cast doubt on the effectiveness of staggering. They provide evidence that staggering has a negligible, or even negative, impact on multiple object tracking tasks [56]. This could be a result of a loss of common fate (i.e., when objects move at the same velocity along parallel trajectories) as each object moves after the other object stops.

Chalbi et al. reveal that graphical perception is improved by common fate. This evidence indicates that staggering with an overlap could mitigate the common fate loss as animations are grouped together rather than animate in isolation. Related data storytelling tools support staggering as it is believed to prevent occlusion and crowding in cases of

structured data and certain motion paths [33, 32]. However, further evidence is needed to prove the benefits of staggering.

Trajectory bundling is a related method to group similar animations together and remove occlusion. Evidence shows that this method is particularly effective when tracking multiple targets [57]. However, trajectory bundling relies on altering the interpolation path of an animation rather than temporal pacing. When considering which easing function to use, Dragicevic et al. determined that "slow-in/slow-out" typically outperforms other functions for point clouds yet more research is needed for other visualization types [58].

Additional findings from empirical research and systems research has produced guidelines for when and how to use animation for visualization. Tverskey et al. use the congruence and apprehension principles as guidance for assessing whether or not animation can facilitate similar comprehension and communication as static graphic representations [59]. They conclude that animation provides no benefit for communicating complex processes (e.g., algorithms). However, they make an exception for animated transitions in cases where the transition can be "accurately perceived and appropriately conceived" by viewers. Heer and Robertson build upon the congruence and apprehension principles with actionable guidelines for animation in statistical data graphics [54]. Their guidelines include: "maintain valid data graphics during transitions", "group similar transitions", "minimize occlusion", "use simple transitions", and "use staging for complex transitions." Informed by the design and implementation of a timeline storytelling tool, Brehmer et al. propose guidelines for animated transitions between timelines of varying representation, scale, and layout [32]. They encourage the use of staging in cases of highly salient changes, staggering of object groups during translation and scaling transitions, and simple transitions that involve at most one dimension change at a time. Our approach aims to support designers to author animated data graphics that follow the guidelines put forth by previous work.

2.2 Authoring Tools for Static Data Graphics

Here I summarize the authoring tools from research and industry that support generative data binding and encoding features for static graphics. Rost [60] has previously surveyed 24 visualization toolkits and interactive systems by comparing her experience re-creating a static version of Rosling's iconic GapMinder visualization [61] with each tool. I draw on Rost's survey to identify visualization authoring tools from industry. Grammel et al. [62] also provides a comprehensive survey of visualization authoring systems. I refer to Grammel et al.'s categorization of interactive systems as *template editors*, *shelf construction*, and *visual builders* in the sections below.

2.2.1 Programming Toolkits and Declarative Grammars

Graphical grammars [63, 64] have served as the basis for powerful and expressive data visualization toolkits such as ggplot and ggplot2 [11, 65]. The grammars from Wilkinson [63] and Wickham [64] both follow a bottom-up approach: starting from data, aesthetic mappings from variables and coordinate systems drive the visual form of the graphic. Our framework assumes sketched shapes instead of data as the starting point. Data is only incorporated into graphics as necessary.

Declarative languages [66] provide a higher-level abstraction for constructing interactive visualizations by de-coupling specification from execution. Visualization toolkits built along this research direction [67, 68, 10, 69] simplify the construction of visualizations while preserving a broad design space. D^3 in particular provides powerful capabilities in an accessible form, while supporting design freedom beyond data-driven mappings. Designers can create annotations, visual embellishments and additional structure alongside the declarative data mappings supported by D^3. The popularity of D^3 has been attributed to its re-use of an ubiquitous medium: the Document Object Model (DOM). Our approach strives to mimic this paradigm, as D^3 enables direct manipulation of the DOM, we augment

vector editing tools to interact with the scene graph. While D^3 requires designers to select, bind, and style DOM elements by writing JavaScript code, We provide a visual and direct manipulation approach for non-programmers. Declarative grammars still pose a challenge to designers, as they must write textual code instead of designing visually.

2.2.2 Template Editors

Template editors provide a set of charts for authors to choose from. Examples include community driven systems like Many Eyes [70]; tools for data journalists like DataWrapper [15]; systems with an infographic focus such as Infogram [71], PiktoChart [72], Easel.ly [73], InfoNice [74]; and many more: Flourish [14], RAWGraphs [17], Plot.ly [16], Datamatic [75], Quadrigram [76].

The broader public frequently interacts with template editors in tools such as Microsoft Excel [77] – the system presents templates based on a selection of data columns and rows in the current worksheet. Template-based approaches can also be found in tools such as Microsoft PowerBI [78] and Google Data Studio [79]. The template interaction paradigm has a low-threshold for authoring visualizations; it appeals to novices and time-constrained experts alike. Several template editors (e.g., RAWGraphs [17], Flourish [14], PowerBI [78]) provide Application Programming Interfaces (APIs), allowing developers to create new templates. These tools quickly generate charts for users to compare design alternatives, however users are restricted to a predefined set of chart types and only a handful of customization options.

2.2.3 Shelf Construction Interfaces

Shelf construction interfaces expand upon the expressive limitations of template editors. Authors construct visualizations by mapping data fields to visual encoding shelfs (e.g., position, color, shape). The system relies on a framework to generate a valid chart given the author's shelf specification. Examples include commercial tools like Tableau [18] (previ-

ously Polaris [80]); and research systems like Voyager [81], Voyager 2 [82]. These systems do not provide control over the underlying chart layout, nor do they allow authors to easily produce compound glyphs comprised of multiple marks. Tableau is based on a table algebra framework [80], where operators such as cross and nest work solely on data. Visual marks only appear in a later stage of the pipeline. In the Data Illustrator framework, operators such as *repeat* and *partition* primarily work on visual components, with data as ingredients. We base our approach on the assumption that designers want to draw first, then incorporate data into a design. Shelf construction interfaces prompt users to start by interacting with data attributes. Tableau also supports detail-oriented customizations on scales and visual configurations, but these customizations often have to be accomplished through dialogs. Our approach brings the customization of scales, axes, and legends directly to the canvas with direct manipulation widgets.

2.2.4 Visual Builders

My work continues the line of research that explores visualization composition using graphical primitives. In early research such as SageBrush [83], users choose chart type from a list of prototypes, add graphemes to the prototype, and specify mappings between data and grapheme properties. SketchStory [84] and SketchInsight [85] use freeform shapes as archetypes to be repeated and transformed with data mappings. Victor [21] and Schachman [86] contribute procedural methods for designers to create parametrically generated graphics.

Additional systems have investigated novel interaction techniques for visualization authoring. Early research explored programming by demonstration as a method for creating charts and described heuristics for inferring user intention in chart specification [87]. iVoLVER [24] supports the extraction, transformation, and presentation of information using pipeline style widgets in the canvas. Our approach takes interaction design inspiration from the progressive disclosure techniques used in the iVoLVER system.

Data-Driven Guides (DDG) [25] and DataInk [26] take a similar approach to ours by augmenting existing drawing tools. DDG treats marks as flexible, deformable graphical elements, while DataInk supports pen-an-touch interactions to draw glyphs and hand-drawn layouts. Data Illustrator supports a wider range of data-to-visual mappings and more complex layouts than DDG and DataInk. Related to our approach for layouts, Vuillemot & Boy [88] define a framework to assist designers in creating visualization mock-ups by employing top-down approach for subdividing the scene graph. The Data Illustrator framework is similar to the segmenting, nesting, and linking portions of their framework. Our approach is capable of high-fidelity visualizations instead of mock-ups.

Recent data visualization research has sought to empower designers to create expressive visualizations without the need to program. Tools such as Lyra [23] and iVisDesigner [22] aim to provide users with the power of declarative toolkits in a familiar vector editing interface. Both systems employ higher-level representations of the scene graph: Lyra is built upon the Vega visualization grammar [89], while iVisDesigner's custom framework supports templated plots. User interaction modifies the abstraction, which in turn updates visualization rendering. In our approach, we do not have such abstractions: user interaction directly translates to operations on the scene graph. This choice allows us to focus on the interface and interaction design first, while the system architecture and visualization model come second.

Recently, Ren et al. introduced Charticulator [27], a constraint-based visualization authoring system. Charticulator excels at authoring complex data glyphs, with a constraint-based approach for relative positioning. However, our approach allows glyphs to represent one or more data tuples. Charticulator outperforms Data Illustrator's layout capabilities by supporting polar, circle-packing, and non-linear layouts. The user experience for selecting and manipulating marks in Charticulator is in-direct as constraints are incrementally adjusted based on user adjustments. Our approach, on the other hand, provides direct manipulation of marks and lazy data-bindings. Our approach for direct selection and manipulation

is based on an understanding of graphic designers' desire for control. For more details on how Data Illustrator, Lyra [23], and Charticulator [27] compare and contrast, refer to our co-authored paper that critically reflects on these visualization authoring tools [90].

2.3 Authoring Tools for Animated Data Graphics

Authoring animation in data graphics requires building on top of the previously discussed capabilities for authoring static data graphics in Section 2.2. In this section I highlight which of those tools support animation. Animation introduces additional complexities and considerations to authoring tools, such as coordinating the relationship between data graphics, constructing a narrative, or specifying animation designs. Here I summarize which of the authoring tools from research and industry support animated data graphics. First, I introduce the frameworks and taxonomies that guide our understanding of animation's form and role in data graphics. Then I discuss the authoring tools that support animation, delineated by textual programming and interactive systems.

2.3.1 Frameworks and Taxonomies

Many taxonomies exist to understand the different types of changes in animated data graphics. Heer and Robertson defined a taxonomy of animated transitions in statistical data graphics [54]. Their taxonomy defines 7 types of animated transitions by considering the syntactic or semantic operators one might apply to a visualization. Fisher [53] adapted this taxonomy and proposed a list of six animation types in visualization. DataClips [91] identified high-level building blocks of data videos expressed in visualization type $\times$ animation type combinations. Chalbi [92] distinguished between data-driven changes and visual-driven changes, and enumerated animated changes at the level of low-level components. Chevalier et al. [93] went beyond data graphics to examine the different roles played in animation in user interfaces, so that novel uses of animation and research opportunities could be identified. Similarly, our approach identifies the primitives to be used in the au-

thoring of animated data graphics along four dimensions: object, graphics, data and timing. We then propose exemplar compositions of these primitives, such as transition types and pacing techniques that informed the aspects of the Data Animator framework.

2.3.2 Programming Toolkits and Declarative Grammars

Programming toolkits and visualization grammars have led the way in animation for visualization. Imperative programming toolkits [12, 94] update graphics in a step-wise manner on each frame update. Our approach is more closely related to declarative programming approach in which graphics animate between declared visual states without worrying about the minutia of drawing each frame. Declarative grammars provide further abstraction from the low-level details needed to create interactive visualizations [95, 96]. Similar to how the graphics rendering of Data Animator leverages the GPU for performance, Ren et al. also provide access to GPU-enhanced rendering in a familiar declarative programming library for producing visualizations [97].

The gganimate [98] library, an extension of ggplot2 [65], builds on a foundation grammar [64] with its own "animation grammar" that tweens different components of the data to graphic pipeline (e.g., data, aesthetic mappings, coordinate systems). This approach focuses on animating a single, isolated data graphic, rather than creating animated transitions between two visualizations. More recently, Tong et al. introduced a high-level domain-specific language (DSL) that enables declarative specifications of chart animations by leveraging data-enriched SVG charts [99]. Canis supports applying animation effects and temporal functions to selected marks or groups of marks. However, programmers must rely on carefully formatting data-enriched SVG files in order to coordinate matching objects between each file. Kim and Heer [100] introduce a declarative grammar for specifying transition steps and build a recommender system to assist designers. Our research goal is to help designers define transition steps and additional pacing methods through a graphical user interface, and automated recommendation of animation design is not within the scope

of that work.

D^3 [10] is ubiquitously used to create animated data narratives by data journalists, bloggers, and designers. Leveraging the Document Object Model (DOM) for interaction and graphics rendering, D^3 provides a flexible approach for programming visually diverse data presentations. D^3's "transition" module animates selected DOM elements from their initial visual properties to newly declared visual properties. Beyond animated transitions, D^3 also offers modules that assist in state-based animations that are typically reserved for imperative programming languages (e.g., force-directed simulations). This broad range of animation would not be possible without D^3's enter, update, exit paradigm to manage the differences between current and next visual state of a data graphic. We model the Data Animator concepts for matching objects between visualizations after the enter, update, exit paradigm. In addition, D^3 supports pacing selected elements with declarative functions to vary delay, duration, and easing based on data. However, D^3 still poses a challenge to designers, as they must write textual code instead of designing visually. Writing, compiling, and re-running textual code is time-consuming and difficult to learn for designers. This workflow contrasts sharply with how designers typically preview animations in design tools with full playback functionality. Designers often favor familiar design tools for animation that provide design freedom, direct manipulation, and rapid feedback.

2.3.3 Interactive Tools

When it comes to incorporating animation into data visualizations, the number of interactive authoring tools are few, far between and outpaced by the generative capabilities of textual programming. All the interactive systems that support animation are template editors [15, 101, 17, 14]: designers choose a visualization from a limited set of designs, and further add or customize through a dialog interface. In contrast, Data Animator would be first to introduce animation into a visual builder approach. Amini et al. introduced DataClips, a template system for creating data videos by sequencing clips. Clips are com-

binations of visualization type $\times$ animation type [33]. Unlike Data Animator, DataClips does not support animated transitions between two visualizations. The Microsoft platform supports animation by coupling charts and tables created in Microsoft Excel to the animation effects of PowerPoint [102]. Additional domain-specific tools support animated transitions between timeline visualizations [103, 32] and node-link graph layouts [104]. Data Animator addresses animation authoring for general domain purposes.

Most related to our approach, Flourish Studio allows users to create animated data narratives by sequencing visualizations in a slideshow interface [14]. Flourish Studio also supports templated "data update" animations such as bar chart races, zoomable hierarchies, and globe geo-connections. However, these systems trade off expediency for expressivity, as they help designers quickly create animated data stories yet restrict designs to a predetermined lexicon. The template approach addresses the complexities of animated data graphics by constraining the problem. With limited visualization types, coordinating behavior of visual objects between a transition can be pre-programmed, while support for pacing temporal rhythms is set to inflexible defaults. Our approach seeks to address these considerations in a broader design space for expressive animated data graphics.

2.4 Graphic Design Practices

When it comes to visualization authoring, designers often opt for familiar design tools that fit within their practice. Here I summarize user research to understand graphic design practices when incorporating data. I then go on to summarize interactive systems from industry and research that support designing static vector graphics and animated graphics. These tools provide designers with expressive control over static and animated graphics yet lack support for data-driven generation.

2.4.1 Studies on Visualization Design Process

Understanding how designers approach the visualization design process without writing textual code was crucial for our approach. We built our understanding on top of previous user research that investigates how designers' approach visualization construction tasks. For example, Bigelow et al. analyzed the visualization design approach of graphic designers in [8], highlighting flexibility over generative capabilities. Mendez et al. reconcile this tension by proposing a "bottom-up" approach for visualization design, as opposed to the "top-down" approach seen in other visualization design systems [13, 105]. Our approach is similar to "bottom-up", as designers construct visualizations from graphic primitives and introduce hierarchy with data-binding operations. While data sketching offers an unrestricted approach to represent data by physically drawing visualizations that span in fidelity: from throw-away sketches to finished compositions [106, 107]. Amini et al. analyze how designers create data stories by observing them create storyboards based on data facts [91]. Our research builds on this work by investigating how designers conceptually approach authoring expressive static and animated data graphics. To inform Data Illustrator, we gathered mockups and storyboards from a design study with three designers. In the case of Data Animator, we conducted an ideation study with 14 designers to understand their authoring preferences and to illicit new interface and interaction concepts. These studies provided us with insights into how we could augment graphic design tools for visualization authoring.

2.4.2 Vector Graphic Design Tools

Professional vector editors enable designers to work with shape geometries at the level of anchor points and curve segments. These vector editing tools also support grid systems, smart guides, and symbols to build repeated elements in a layout (e.g., Adobe Illustrator [19], Adobe Xd [30], Sketch [20]). These features, among others, grant the level of control and flexibility required for designers to create sophisticated data visualizations.

However, these tools have little to no support for incorporating data. Often, designers must hand-encode visualization rulers or compute data to visual encodings using spreadsheets to represent data [25]. This process is time consuming and error prone. Furthermore, design tools only produce rigid, one-off designs; changes in data or visual mappings have to be manually updated.

Professional design tools like Adobe Illustrator [19] provide features such as the *Blend Tool* to duplicate shapes and layers. These efforts inspire our work and suggest the need of a visual language that describes the composition and generation of diverse visualizations. Closely related to our approach for generating and controlling repeated graphical marks is Para, a constraint-based drawing tool for procedural art [108]. Para allows designers to directly generate and layout copied shapes using the *linear, normal, and radial distributions* as interactive constructs, similar to Data Illustrator's *Repeat Grid* and *Partition Stack*.

2.4.3 Animation Design Tools

There are three dominant animation authoring paradigms that are relevant for animated data graphics: *keyframe animation*, *procedural animation*, and *presets & templates*. We define these authoring paradigms for animation as follows:

- *keyframe animation*: specify properties of graphical objects at certain points of time by setting a set of keyframes, frames in between two keyframes are generated by tweening.
- *procedural animation*: generate animation of large number of animated objects with a set of behavior parameters.
- *presets & templates*: apply predefined animation effects and configurations to objects.

An authoring paradigm can be implemented in different input modalities and interaction techniques (e.g., natural language interfaces, programming languages, Graphical User

Interfaces (GUI), sketching, gesturing). Next, I review related research and existing commercial systems that adopt keyframing, procedural animation, and presets and templates.

Keyframe Animation – Motion graphics tools like Adobe After Effects [29] allow users to define properties (e.g., position, scale, opacity) of graphical objects at specific points in time in the form of a keyframe. Intermediate frames are created by tweening the visual properties between defined keyframes. The graphical user interface for keyframing is usually in the form of a timeline editor. We base our approach for Data Animator on the keyframe animation paradigm. Data Animator's timeline editor augments familiar keyframe animation interfaces such as Adobe After Effects [29], Invision [31] and TumultHype [109]. Keyframes' temporal positions are directly manipulated through click-and-drag interactions in Data Animator. Our approach augments the traditional timeline editor for manipulating the keyframes for groups and hierarchies of objects to support data-driven pacing, and relates to the Repeater [110] and Blend [111] plugins for Adobe After Effects which provide mechanisms to generate shapes and control timing *en masse*.

User interface prototyping tools (e.g., Tumult Hype [109], InVision [31], Principle [112], Adobe Xd [30]) support the definition of screens, views, and states of a prototype as keyframes. Instead of defining behavior explicitly on a timeline, sometimes simpler interfaces are used (e.g., users draw a connector to link two art-boards representing two keyframes). These tools combine features of vector graphic drawing tools with capabilities to design the flow of a multi-screen application. Prototyping tools and presentation tools include features that automatically create key-frames based on objects' properties in each art-board or slide, such as "Magic Move" in Keynote [113], "Auto-Animate" in Adobe Xd [30], or "Advanced Animation" InVision [31] . This is similar to our approach for quickly creating animated transitions by connecting two static visualizations in Data Animator's storyboard interface. In storyboarding interfaces, users define transitions between screens, views, or slides by matching objects between them. However, current tools lack the ability to visualize and disambiguate object matches between views. Data Animator introduces a

novel interface to clarify such ambiguity.

Procedural Animation – Procedural animation is a popular paradigm for animations involving many objects, such as particle systems (e.g., fire, rainfall), flocking (e.g., a school of fish), and stochastic motion (e.g., leaves in the wind) [114, 115]. These animating behaviors are typically created procedurally: starting with an initial condition, an engine (often called an "emitter" or "oscillator") generates motion by adjusting a set of parameters (e.g., wind strength, object's stiffness). Our approach does not support procedural animation – as the imperative model of procedural animation is at odds with the declarative structure of keyframing. While direct manipulation interfaces have been proven to be effective in animating procedural illustrations [115, 114, 21], they are ill-suited for transitioning between two visual states and can produce unexpected results.

Presets & Templates – For novice and casual users, it is not always beneficial to use a powerful tool that forces them to start from scratch. Reusable animation presets & templates can significantly lower the learning threshold and reduce the time and effort needed. These predefined animation types are sufficient in many use cases. Presentation tools [113, 102] offer animation effects both at the frame level (e.g., slide transition effects) and at the object level in presentation slides (e.g., "Fade In", "Move In", "Wipe In"). Data Animator's framework borrows many of these preset animation effects such as "Fade Out" and "Fade In" for adding animation to exiting and entering visual objects. Previous visualization authoring tools [14, 33] wholly adopt presets & templates as they reduce the time and effort needed to create animations. While Data Animator is not based on presets & templates, we selectively introduce templates to hide the low-level details for complex functions such as data-driven pacing.

CHAPTER 3

DATA ILLUSTRATOR

In this chapter, I address research questions RQ1, RQ2, & RQ3 for authoring static data graphics. I elaborate on our approach for supporting graphic designers to author expressive static data graphics. I will cover the formative work that informs our framework – highlighted by analysis of visualizations from the web, and a two-year design study with three designers. The subsequent Data Illustrator framework satisfies RQ1. Our framework takes a graphics-first approach: data is incorporated into vector graphics through data binding operations and expressed via lazy data-bindings which only constrain interactive manipulation to that data bound property. The framework augments graphic design tools with new concepts and operators, and describes the structure and generation of a variety of visualizations. To address RQ2 we design and implemented the Data Illustrator system, based on the framework. The system extends interaction techniques in modern vector design tools for direct manipulation of visualization configurations and parameters. In response to RQ3 we report on a re-creation study that shows designers can understand and use our framework to author visualizations. This work is published as a conference poster [116] and a conference paper [35].

Contributors – The completed work in this chapter has been equally contributed between Zhicheng Liu of the University of Maryland (previously Adobe Research) and myself. Zhicheng Liu and I collaboratively completed the formative work, the framework, system design and implementation, user studies, and deployment of Data Illustrator. Alan Wilson, Bernard Kerr, Sam Grigg, and James Delorey of Adobe Systems contributed their own designs during our participatory design exercises over a period of two years. Alan Wilson, Sam Grigg, and James Delorey also assisted in the generation of supplemental materials for

the system's deployment. Mira Dontcheva of Adobe Research and John Stasko of Georgia Tech provided advisory assistance and guidance on this completed research.

3.1 Introduction

Graphic designers have been producing infographics and charts well before the recent proliferation of computer generated visualizations [117, 118]. As visualization becomes an increasingly popular medium for storytelling and communication, there is a renewed and growing interest to understand visualization creation from the perspective of graphic design [8, 119, 25, 88, 106]. Prior studies show that graphic designers approach visualization authoring differently from computer scientists: they often start by thinking about the high-level appearance of a visualization in terms of layout and space configuration, and focus on encoding real data into the visuals later [8, 88]. The discipline of graphic design has also established a rich set of concepts and tools that are widely used in the community. For example, professional vector editors enable designers to work with shape geometries at the level of anchor points and curve segments. The grid system and smart guides serve as two powerful tools to precisely structure visual elements and configure display space [25, 120, 88].

Despite the plethora of existing visualization creation tools, few tried to incorporate designers' workflow and practices into system and interface design. Vuillemot and Boy [88] argue that most visualization tools follow a bottom-up, data-to-graphics process as described in the information visualization reference model [121]: starting with data, one performs data transformation, visual mapping, and view transformation to generate visualizations. This model informed the development of powerful visualization algebra and declarative languages [89, 96, 80, 65]. However, these tools often require coding expertise, or are not flexible enough for design practices.

Systems like Lyra [23] and iVisDesigner [22] offer graphical user interfaces (GUI) for visualization authoring without programming, thus are more flexible. These efforts start

with template or grammar-based visualization generation engines, and design interfaces for changing generative parameters. Such approaches still need to reconcile the potential tension between flexible change of graphical configurations and the formalism imposed by generation engines [119]. To bridge the gap between generation engines and drawing tools, Hanpuku [119] implements a streamlined model for visualization authoring across multiple tools.

Recent work also began to explore visualization authoring without programming from a purely graphic design perspective. With Data-Driven Guides [25], designers can create freeform guides and sketch graphics with the guides. d3-gridding [88] enables the creation of quick mock-ups with minimal or no data. These systems adopt a "lazy data binding" approach: visualizations are first and foremost vector graphics with no underlying templates or declarative languages. Designers use familiar tools to draw, select and manipulate vector graphics, and apply data encoding only when it is necessary. Compared to template or grammar based systems, this approach is more compatible with designers' workflows and practices. Users do not have to align their mental models with the grammar or model assumed by the system. Furthermore, vector design tools are highly flexible and expressive: with enough time and patience, one can create virtually any graphics. Augmenting these tools with data encoding support can reduce manual effort without disrupting designers' workflows.

The lazy data binding approach is promising, but needs to be developed further to support a wide variety of visualizations. Data-Driven Guides [25] only focus on infographics with simple layouts. d3-gridding [88] primarily supports design mockups, and still requires programming. It remains a challenge for designers to create high-fidelity data visualizations with complex visual mappings and layouts.

Consider the visualizations in Figure 3.1: Figure 3.1.a is a slope graph used on the cover of Alberto Cairo's book *The Functional Art* [9], showing U.S. states' obesity and education percentages (hereafter referred as the "Obesity vs. Education" visualization); Figure 3.1.b

visualizes the NBA draft over the past 20 years (x axis) and the order of players in terms of draft pick (y axis) [122] (hereafter referred as the "NBA Redraft" visualization); Figure 3.1.c is a multi-series line graph visualizing four companies' monthly stock prices [123] (hereafter referred as the "Stock Prices" visualization); Figure 3.1.d is "A Field Guide to Red and Blue America" by Wall Street Journal, showing the PVI (Partisan Voter Index) for each state over the past 9 elections [124] (hereafter referred as the "Red and Blue America" visualization). Each small bar chart represents a state, and is positioned according to US geography.

Designers might be able to use existing drawing tools or Data-Driven Guides to create these examples, but the process will be painful. Generating shapes or points on lines (Figure 3.1.c) can be tedious and slow; organizing the shapes into meaningful layouts (Figure 3.1.b and d) and map data to positions and color (Figure 3.1.c) are daunting manual tasks. To enable designers to keep using the powerful drawing tools and to automate the repetitive work, we need a systematic framework with sufficient descriptive and generative power.

In this chapter we propose a novel framework for visualization authoring based on the lazy data encoding approach. This framework describes components in a visualization using graphic design concepts such as *shape*, *anchor point*, *segment*, and *group*. Two operators, *repeat* and *partition*, generate shapes and anchor points, and attach data to them. The resultant visual components each has a *data scope*, and are considered *peers* of each other inside a *collection*. Collections use *layouts* to arrange shapes, and can be *nested* to create more complex organizations. Data serves as *constraints* when bound to visual properties, and unbound properties can be freely manipulated. These components and operators can describe the structure and generation of a wide range of visualizations.

Informed by the framework, we design and implement the Data Illustrator system. We augment interactive techniques in modern vector design tools for direct manipulation of visualization configurations and parameters. We demonstrate the expressive power of our approach through a range of examples. To better understand the strengths and limitations of

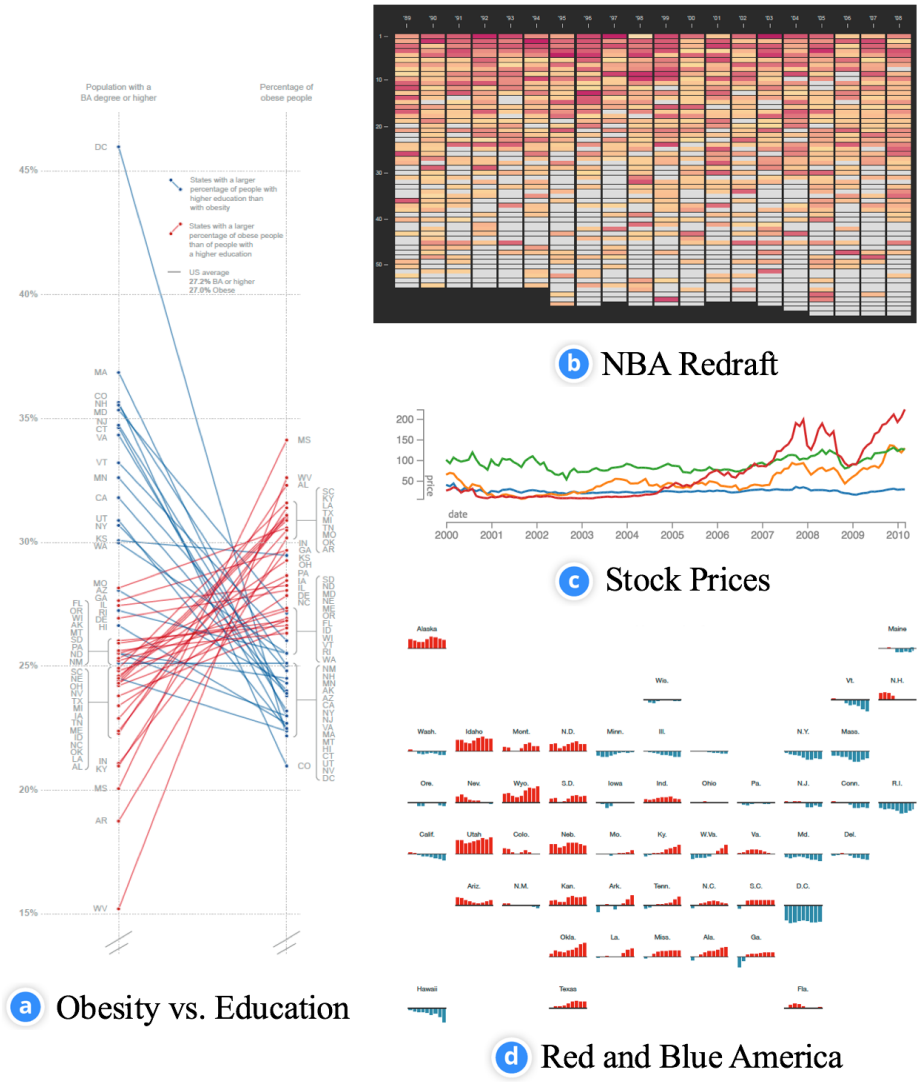


Figure 3.1: Visualizations with varying levels of complexity

our approach, we conduct a qualitative user study with 13 designers, focusing on whether they can understand and use the framework for visualization composition.

3.2 Framework

3.2.1 Formative Research

Deconstructing Expressive Visualizations – We deconstructed a set of creative visualizations into the data elements and data attributes conveyed in the visualization [116]. Fur-

thermore, we categorized each visual mark and attribute driven by data; based on the transformation and encoding employed by the designer. By surveying and analyzing expressive data graphics, this helps to define the graphic primivites and visual encodings that should be supported in our framework.

Our analysis covered 37 examples that showcase a wide variety of data visualization for public or casual use. These visualizations exemplify designs that are innovative, appealing, beautiful and informative. By focusing on this genre of visualizations, our analysis aims to expose a visual grammar that either conforms, evolves, or deviates from the canon of visual grammars. The visualizations are taken the Kantar Information Is Beautiful Awards of 2014 [3].

The results of the analysis is a total of 761 visual encodings and a total of 271 data objects. The product of our analysis, the meta-data from these annotations, exposes detailed information on each encoding. This meta-data helped in our synthesis of a framework for expressive data graphics. In particular we identified styling techniques and variations in encodings that separate these visualizations from more generic charts and plots (e.g., position-encoding of line segments/anchor points, freeform vector shapes as data marks).

Participatory Design – To understand how different visualizations could be described and created from a graphic design perspective, we held one-hour weekly meetings with three designers over a period of two years. All three designers have more than ten years of experience in graphic design, digital illustration, web design and print design. Two of the designers have also created infographics and data visualizations on a regular basis as part of their work. The designers frequently used applications such as Adobe Photoshop [125], Illustrator [19], InDesign [126] and Adobe Xd [30], Sketch [20], and Figma [127]. These applications represent the industry standard for design professionals. They share a similar set of features and tools, varying in terms of interaction and interface design.

In the initial meetings, we collected visualizations by sampling chart types from systems like Tableau [18] and stylistic information graphics from websites such as “the Kantar

Information is Beautiful Awards" [3]. Each week we asked the designers to describe at a high level how they would create one of these visualizations and demonstrate the workflow using the tools of their choice. We told the designers to assume that the system would take care of data binding automatically. These exercises helped us understand designers' way of thinking and workflow through concrete examples, and familiarized ourselves with an assortment of professional design tools.

Three main tasks are key in visualization authoring. Designers performed the following three tasks for all the examples: (1) sketch and generate shapes, which were accomplished by drawing tools (e.g., *Pen Tool* in Adobe XD) and duplication tools (e.g., *Copy & Paste*), (2) arrange and organize shapes, where grids and groups were extensively used, and (3) bind data to visual properties, which was not supported in most design tools. Two pieces of insight were consistent with previous findings [116], and directly informed the focus on *repeat* and *layout* in our framework: repeated shapes were dominant in data visualizations, and position encoding were often relational instead of data-driven (i.e. a shape's position depended on the placement of related shapes).

Workflow is largely top-down, but data is not always an afterthought. Observations from these exercises confirmed our intuition and previous findings [8, 88] that designers think about graphical aspects of visualizations before data encoding. However, the authoring processes were not strictly divided into a visual design stage followed by a data encoding stage. When drawing and manipulating shapes, sometimes it was beneficial to bring in real data. For example, one designer showed how he would use the *Blend Tool* [128] in Illustrator to create multiple copies of a shape. He first drew two shapes on canvas, and then used the *Blend Tool* to interpolate a predefined number of shapes between them. Instead of having to define an arbitrary number, the designer wanted automatic generation of the number based on real data.

Direct manipulation enhances flexibility and reduces semantic distance. Designers treat the canvas not only as a scene for production, but also a playground for experimenting with

ideas. It is important that they can flexibly and quickly sketch shapes, and change visual configurations or appearances with full control and precision. All the designers greatly valued direct manipulation features that gave them immediate visual feedback. Simple operations such as dragging corners to resize a shape, or dragging shapes to move them around can be immensely useful. Such features are commonplace in design tools, but are rarely supported in visualization systems.

In the second phase of the formative study, we explored how existing design tools could be directly used or augmented to support the three main tasks. We conducted weekly design meetings for 15 months. Each week we created a storyboard to illustrate step-by-step visualization authoring scenarios. In total we produced about 40 design sketches and mockups. For instance, we spent one month brainstorming how to augment existing design tools to create visualizations in the line graph category. In a line graph, one poly-line plots all the data, and the points on the line represent individual data cases. Grammar-based approaches solve this problem through declarative specification (e.g., *line*(*position*(date $*$ value)) [63]) . To designers, however, such specification did not make sense because a line's position refers to the coordinates of its bounds. Taking a graphics-centric approach, we created storyboards based on different ideas such as repeating points along a line, duplicating a point multiple times and connecting the dots, and dividing a line into segments. We then collected designers' feedback and eliminated ideas that sounded bizarre to them.

A great challenge we faced was to construct a coherent set of concepts and tools that behave consistently for diverse visualizations. Often an idea seemed feasible for one visualization design, but turned out to be inconsistent with new examples. In addition to a single line graph, we needed to consider more complex cases such as small multiples of line graphs or multiple lines in a single chart. Moreover, slope graphs and parallel coordinate plots also use line as a visual primitive, but in different ways from a line graph. These diverse visualizations added complexity to the construction of a coherent framework. We kept iterating on the ideas as new use cases arose. After a few months of storyboard

creation, we distilled a set of concepts and tools that worked consistently across different visualizations. We then started implementing a prototype which helped further solidify the ideas, and the iterations on framework and interaction designs continued towards the end of the prototyping process.

3.2.2 Framework Components

The principle of consistency underpins the creation of our framework. If we borrow an existing design concept, its meaning and behavior must be consistent with the way it is used in existing design applications. Otherwise we need to devise a new concept. For example, *symbol* is widely adopted in mainstream vector design tools. Users can turn any visual object into a symbol (akin to the concept of class in computer science), and create many instances of the symbol. Changes to the symbol will be propagated to the instances. We tried to use *symbol* to describe the generation of visualizations, but eventually decided that it was a stretch to apply it in the context of visualization authoring. To explain the framework, we use the visualizations in Figure 3.1 as running examples.

Shapes, Anchor Points and Segments – Shapes are the building blocks of visualizations. In professional design tools, shapes are represented as series of anchor points connected by line or curve segments. Table 3.1 shows a few shape types, with information on the number of anchor points, the number of segments, and whether the path is open or closed. A line is the shape primitive for Figure 3.1(a), a rectangle for Figure 3.1(b) and (d), and a polyline/path for Figure 3.1(c).

Designers sketch the shapes with drawing tools for different shape types (e.g., *Rectangle Tool*, *Ellipse Tool*, and *Pen Tool*). They use the *Selection Tool* to select shapes, and the *Direct Selection Tool* to select and manipulate anchor points and segments.

Repeat and Partition – After sketching a shape primitive, designers can use the repeat and partition operators to generate shapes and attach data to them (Table 3.2). Repeat creates

Table 3.1: Anatomy of Shapes: Anchor Points and Segments

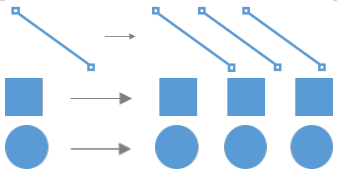
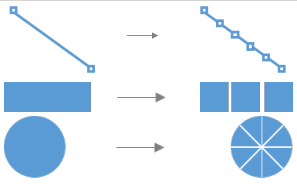
Line	Path	Rectangle	Circle
			
2 anchor points 1 line segment open	4 anchor points 3 line segments open	4 anchor pointss 4 line segments closed	4 anchor points 4 curve segments closed

multiple copies of a shape, and is inspired by duplication tools (e.g., *Repeat Grid* in Adobe XD, *Duplicate* in Sketch); Partition divides a shape into constituent parts, and draws inspirations from the *Knife Tool and Scissors Tool* in Adobe Illustrator.

To create the lines in the "Obesity vs. Education" visualization, we can first repeat a line by State. The repeat operator duplicates the line, and associates each line with a unique State value and the data rows sharing that value (Figure 3.2(a)). Similarly, we can repeat a rectangle by State (Figure 3.2(b)) for the "Red and Blue America" visualization. Note that multiple rows share the same State value, and the repeat operator only generates a shape for each unique State value. In general, the data variable used to repeat a shape should be categorical, since the number of repeated shapes must be an integer.

To create the line graphs in the "Stock Prices" visualization, we follow suit and repeat a line by Company (Figure 3.3(a), top). Next, we partition each line by Date to divide them into multiple line segments. The partition operator generates an anchor point for each Date

Table 3.2: Generative Operators: Repeat vs. Partition

	Repeat	Partition
Concept	creates multiple copies of a shape	divides a shape into constituent parts
Shape	works for all kinds of shape and group	works for lines, rectangles, circles, rings and areas only
Example (line) Example (rectangle) Example (circle)		

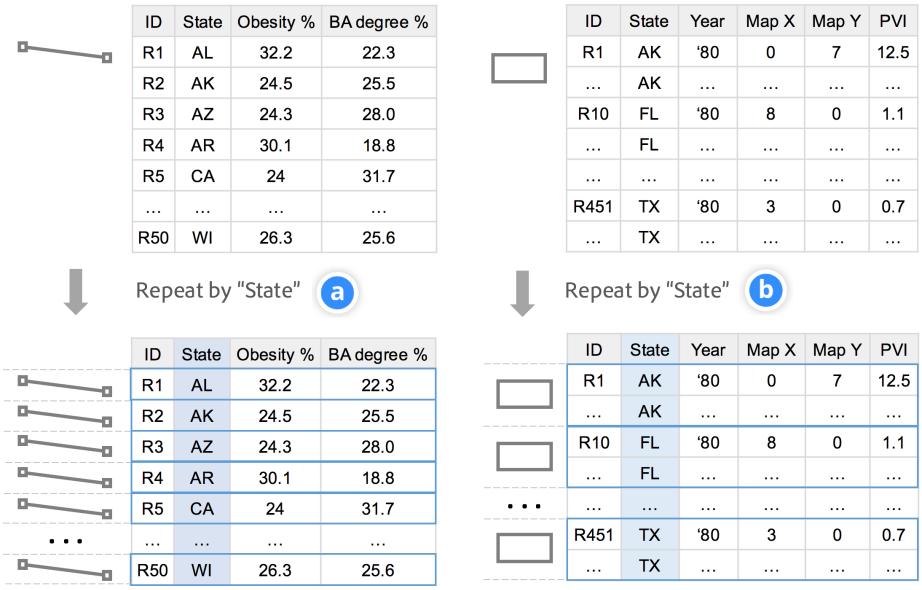


Figure 3.2: (a) repeat a line by State for "Obesity vs. Education", (b) repeat a rectangle by State for "Red and Blue America"

value, and associates the corresponding data rows with the anchor points (Figure 3.3(a)). Similarly, to generate an outline for the "NBA Redraft" visualization, we repeat a rectangle by Year first (Figure 3.3(b), top), then partition the rectangles by Player. In general, the partition operator divides a shape into its constituent parts by a categorical variable. How the division works depends on the shape type, for example, a circle is divided into slices of pie (Table 3.2).

Data Scope – A shape's data scope refers to its attached data rows as a result of the repeat or partition operator. The data scope is usually a subset of the original dataset, described by categorical filters. For example, in Figure 3.3(a), after repeating, the data scope of the first line is the data rows where Company = Microsoft. The anchor points of the line has no data scope yet, only after partitioning, each anchor point has its own data scope: e.g., Company = Microsoft and Date = 01/01/2000. The first filter is inherited from the line's data scope. When we repeat a group by data, all its children share the same data scope. When we partition a line by data, each anchor point has its own data scope.

Figure 3.3: (a) repeat a line by Company then partition lines by Date, (b) repeat a rectangle by Year then partition rectangles by Player

Collection vs. Group – – After repeat (Figure 3.2), we have a collection of lines or rectangles for each of the four examples in Figure 3.1. This collection may be considered as a "group": in all the design applications, multiple shapes can be grouped so that they can be moved, scaled or copied at once. In our framework, however, collection and group are two distinct concepts.

Layout – To arrange the lines and rectangles in a collection, we apply a layout to the collection. Figure 3.4 shows an example of a grid layout with one column and four rows. The pink lines are the grid cell boundaries. In general, we can apply the following types of layout to a collection generated by repeat: freeform (i.e. no layout), grid, stack, and packing. Grid layout is an essential tool supported by most design applications. Stack layout exists in fewer applications (e.g., Auto-Layout [129] in Sketch), but is an important feature in visualizations. The main differences between a grid layout and a stack layout include: (1) a grid is two dimensional (rows and columns), a stack is one dimensional (horizontal or

vertical); (2) all the grid cells have the same size, and the size of the collection depends on the cell size and number of cells; in a stack layout, the size of the collection is the sum of all the children's sizes. A packing layout is a space-filling arrangement: the layout is equivalent to a Treemap for rectangle shapes, and a packed bubble chart for circles.

Both the grid and stack layouts have a *coordinate space* parameter consisting of two values: Cartesian and Polar. Grid layout in the polar space is inspired by the *Polar Grid Tool* [130] in Adobe Illustrator. Similarly, stack layout has a corresponding representation in the Polar space.

Nested Collection – Collections can be nested to create small multiples or visualizations with nested layouts (e.g., stacked bar chart) [131]. We have seen how to create a nested collection in Figure 3.3(b): first repeat a rectangle by Year to get a collection, then partition the rectangles in the collection by Player. This procedure will generate the structure in the "NBA Redraft" visualization, if we apply a one-row grid layout to the top level collection, and a one-row grid layout to the inner collections obtained from partitioning. Nested collections can also be created by repeating a collection.

Lazy Data Binding as Constraint – By default, the lines or rectangles generated by repeat or partition behave as regular vector graphics. Users can select, scale, move, rotate, align, distribute, and delete the shapes. Even after we have organized these shapes in a collection and their positions are constrained by the layout, we can still move the collection as a whole, or edit the anchor points' position and stroke color. Such flexibility allows manual encoding of shape properties, which could be tedious. Automatic data encoding reduces the manual efforts needed, and serves as additional constraints on the manipulability of visual components.

Say we want the stroke color of the four polylines in Figure 3.4 to represent Company to match the "Stock Prices" visualization. We specify a data binding consisting of four parameters: a data variable (Company), a visual property (Stroke Color), a list of visual

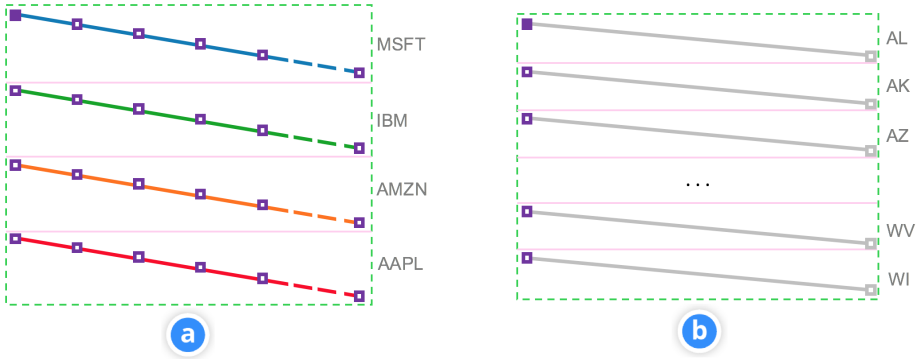


Figure 3.4: Peers of an anchor point: (a) partitioned polylines, each anchor point has a data scope, (b) paths where the anchor points have no data scopes. The focal anchor point is colored in purple, their peers have purple borders.

components([line1, line2, line3, line4]), and an aggregator (e.g., Sum or Mean) if the data variable is numerical and we need to aggregate multiple values. Once applied, the data binding locks the Stroke Color property and prevents it from interactive manipulation. It is still possible to change the range or domain of the scale, which in turn updates the colors. Unconstrained interaction is restored if the data binding is removed.

The data binding operator executes in three steps. First, it computes a list of data values, one per visual component, based on the component's data scope and the aggregator. In the "Stock Prices" visualization, the data values are the four companies. Second, the binding operator creates a scale. The scale type depends on the data variable type and the visual property (the choice of scale type closely follows the guidelines in D^3 [10]); the scale's domain encompasses the data values computed in the previous step; and the scale's range is determined by the visual property values. Finally, the binding operator transforms the list of data value into property values using the scale, and sets the visual properties. Many systems offer automatic data binding support similar to the description above. Our framework differs in the lazy binding as constraint approach.

Peers – Binding Company to Stroke Color results in a unique stroke color for each polyline (Figure 3.4(a)). Next we need to bind Date to the x position and Price to the y position of the anchor points to create the "Stock Prices" visualization. Here we want the data binding

to apply to all the anchor points on all the lines. To create the "Obesity vs. Education" visualization, the data binding works differently. In Figure 3.2(a) we have repeated a line by State, applying a grid layout to the collection gives us the result in Figure 3.4(b). We then need to bind BA Degree % to y position of the first anchor point in each line only, and to bind Obesity % to y position of the last anchor points only.

To distinguish these cases and convey how the data binding will work clearly to the users, we introduce the concept of peers. Shapes generated by repeat or partition are peers of each other. For example, the four polylines in Figure 3.4(a) are peers to each other. What constitutes the peers of an anchor point depends on whether the anchor point has a data scope. When we draw a line and then repeat it by data, the anchor points have no data scopes. The peers of an anchor point are the anchor points at the same index on peer shapes (Figure 3.4(b)). When we partition a line by data, the anchor points are generated and associated with data. All these anchor points are thus considered peers of each other. If we repeat a partitioned line by data, all the anchor points on all the lines are peers of each other (Figure 3.4(a)). The concept of peers helps clarifying which visual components should be affected by a data binding.

Layout Taking Precedence over Position Binding – The structure enforced by layouts sometimes may be in conflict with binding data to positions. In such cases, the layout takes precedence over position binding. For example, with the four polylines arranged in a grid layout (Figure 3.4(a)), after binding data to the positions the anchor points, we obtain Figure 3.5(a). The position binding only takes effect inside each grid cell. Replacing the grid layout with a freeform layout unifies the scales and axes (Figure 3.5(b)).

The framework describes the structure and generation of the backbone of the visualizations. In the actual authoring processes, we still need to perform many lower-level tasks, such as configuring the parameters of a layout, ordering and filtering the collection children, and setting the scale range for data binding. In the next section, we discuss the design of the authoring interface based on this framework, so that we can operationalize the framework

Figure 3.5: (a) The presence of a grid layout has precedence over position binding, (b) Removing the grid layout unifies the scale of position binding

with flexibility and control.

3.3 System

3.3.1 Goals & Design Criteria

We designed the Data Illustrator application with the following design goals in mind: *familiarity, interpretability, discoverability*, and *control*. Realizing these design goals is a crucial step to ensure that our target audience comprehends and enjoys working with a complex authoring tool. *Familiarity* ensures that the user's previous experience will match their expectations; therefore if our tool uses a feature from an existing vector editing application, we want that feature to be consistent in appearance and behavior. In the case when a novel feature is needed, the design should be *interpretable* by the user. *Interpretability* requires the result of a user action to be immediately comprehended, an important requirement during generative data bindings. *Discoverability* on the other hand ensures that the interface design shows affordances for interaction so that users can detect the possibility of an action with a visual object. Finally, users need to feel that they are in *control* at every step of the process to realize their design, especially when the system automatically generates aspects of their work.

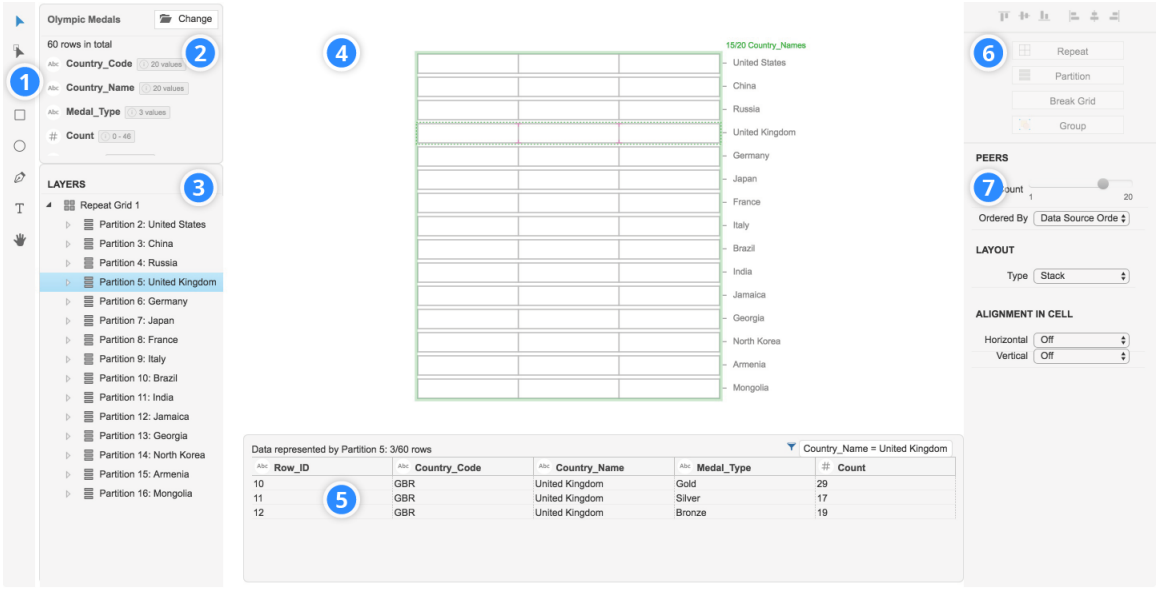


Figure 3.6: Seven components of the Data Illustrator interface: 1) Toolbar, 2) Variables Panel, 3) Layers Panel, 4) Canvas, 5) Table Panel, 6) Actionbar, 7) Property Inspector

3.3.2 User Interface & Interaction Design

The interface of Data Illustrator consists of seven components (Figure 3.6). The Canvas provides space to draw, select and manipulate shapes. The Toolbar on the left contains tools for selecting and drawing shapes - only one can be active at a time. Directly to its right, the Data Variables Panel supports dataset file management. Below that is the Layers Panel which allows users to inspect the canvas. The Actionbar on the far right supports actions for associating data to shapes. Directly below, the Property Inspector displays editable attributes of the currently selected shapes. Finally, the Data Table Panel at the bottom shows all the rows and columns of the dataset and reveals the data scopes of the currently selected shapes.

Drawing Shapes – The drawing tools work by click and drag interactions on the canvas. Data Illustrator supports the following mark types: lines (*Line Tool*), rectangles (*Rectangle Tool*), ellipses (*Ellipse Tool*), text (*Text Tool*), and open or closed non-regular paths (*Pen Tool*). Similar as done in other design applications, the bounding box of the shape remains

active after drawing for further manipulation.

Selecting Visual Components – Selection is a prerequisite for operations such as changing visual properties, transforming objects, associating objects with data, or binding data to attributes. Data Illustrator supports two types of selection: (1) the *Select Tool* works on shapes and collections of shapes, (2) the *Direct Select Tool* works on anchor points and line/curve segments of shapes. The *Direct Select Tool* is a powerful feature in applications such as Illustrator, providing essential control to edit paths and deform regular shapes (e.g., rectangles). Both selection tools use familiar interactions such as: click to single select, shift+click to add to a selection, click+drag to lasso a selection, and clicking on the canvas to deselect. Selection tools are also used to transform objects: click+drag on an object to move it, click+drag on a bounding box corner to re-size a shape, or pressing an arrow key to nudge the selection. The rich selection of interactions in Data Illustrator provides the precise control required by designers.

Working with Data – Data Illustrator allows users to work with one tabular dataset at a time. Users can choose from a spectrum of sample datasets from various sources, or upload a CSV file from their own computer. Upon loading a dataset, the system infers the data types of each column with the Datalib library [132], displays data column summaries in the Data Variables panel, and shows the complete dataset in the Data Table panel. The Data Table also acts as an inspector for the data scopes of the currently selected visual items.

Context-Sensitive Interface – We design the interface to be context-sensitive so that users can understand the possibility of actions at any state. The buttons in the Actionbar are en-

Figure 3.7: Selecting shapes: (a) *Select Tool* selects entire shapes and collections of shapes. (b) *Direct Select Tool* selects anchor points and line segments of shapes.

abled and disabled based on selection on the canvas. For example, if a group is selected, the "Partition" button is disabled, indicating that partitioning a group is not allowed. Similarly, the Property Inspector displays a set of property controls based on the shape type of current selection.

Repeating – Repeat actions begin with the selection of a visual object (i.e. shape, group of shapes, or collection). Clicking the *Repeat* button displays a preview of how the selection will be repeated by a categorical variable. The preview supports changing the categorical variable. Upon confirmation, the repeat action duplicates the selected object, and places the two objects in a default grid layout. We chose to generate only two copies of the object because for large datasets, the number of objects will be overwhelming. To enable designers to control the number of objects to work on, we augment the *Repeat Grid* tool from Adobe XD (Figure3.8.a). Users control the number of generated objects and the grid layout parameters by the following interactions: click+drag handles to display additional rows or columns, click+drag padding to adjust spacing, double-click to open the collection and select objects inside. Dragging past the total shapes allotted by data does not generate further repeated shapes.

Partitioning – Like the repeat action, the partition action requires a selection of a shape with or without a data scope. Groups or collections cannot be partitioned. Clicking the *Partition* button will display a preview of how the selected shape will be divided. Changing the data variable updates the preview auxiliary lines. Partitioning a rectangle results in a stack

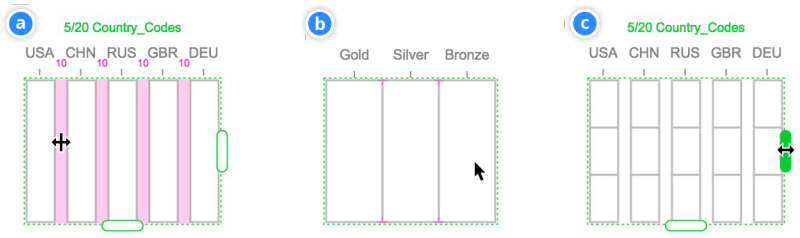


Figure 3.8: Collections with layouts: (a) Repeat Grid. (b) Partition Stack. (c) Partition Stacks Nested in a Repeat Grid

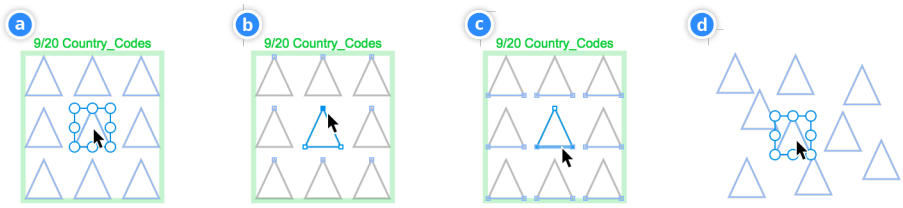


Figure 3.9: Peers highlighted on selection: (a) Peer shapes. (b) Peer anchor points. (c) Peer line segments. (d) Peer shapes in free-form layout.

layout of the sliced rectangles. Unlike the grid layout, the stack layout does not support padding. Nested structures can be created by partitioning a shape multiple times - the data hierarchy limits the number of partition actions.

Peers – Peers are the atomic objects created by repeat and partition actions. Similar to symbols in Sketch [20], or components in Figma [127], peers share visual properties. Upon selecting any shape, Data Illustrator highlights its peers with a faint blue to show linkage between peers. This design applies to anchor points and segments as well (Figure 3.9). Changes to a shape are instantaneously updated to its peer shapes. Properties shared between peers include: appearance, scaling, positioning anchor points or line segments, and data bindings. The only non-linked property is shape position. Grid and stack layouts provide positioning to peers. When the user *breaks* a layout, the peer shapes can be positioned freely.

Breaking a layout is an irreversible action. Removing a grid layout poses a problem for controlling the number of displayed shapes without handles to reveal rows and columns. To remedy the loss of control over peer shape display, Data Illustrator provides a *Peer Count* slider in the Property Inspector.

Lazy Data Bindings – To map visual attributes to data, the user selects any object on the canvas. The Property Inspector populates with the corresponding set of properties. To bind data to property, tools such as Tableau [18] or Lyra [23] let users drag and drop a variable to a property field. We did not choose this design because dragging and dropping

require significant cursor movement, and it is not clear which variables can be mapped to a given property. In our design, the user clicks the binding icon next to the property control, which displays a list of applicable data columns (Figure 3.10 left). Selecting a data column creates a data binding between the underlying data scope and that visual property for all peer objects. For each binding, the system creates a scale, where the range depends on the current values of the visual property. For example, position bindings use the bounds of all peer shapes, and continuous color bindings use the original hue of the selected object.

Data bindings are lazy in Data Illustrator, meaning that they constrain only their bound visual property. For example, position bindings only constrain the position of peer objects in relation to each other. Dragging a position-bound peer object will move the other peers and axis together. A peer shape's data scope may change over time as subsequent repeat or partition actions are applied to peer shapes or collections. Lazy data bindings re-compute to apply the same mappings given the changes in data scopes. Furthermore, the scales behind data bindings can be re-used on other peer shapes to adhere to a cohesive design. If a data binding is removed by the user, peer shapes do not revert back to their previous, unbound appearance. Instead, the shapes return to being standard vector shapes that can be manipulated via drawing interactions like dragging to resize, rotate, or move. Lazy data bindings give designers control within a generative action to manipulate designs as part of a flexible and rich design process. For example, designers can position peer shapes in an

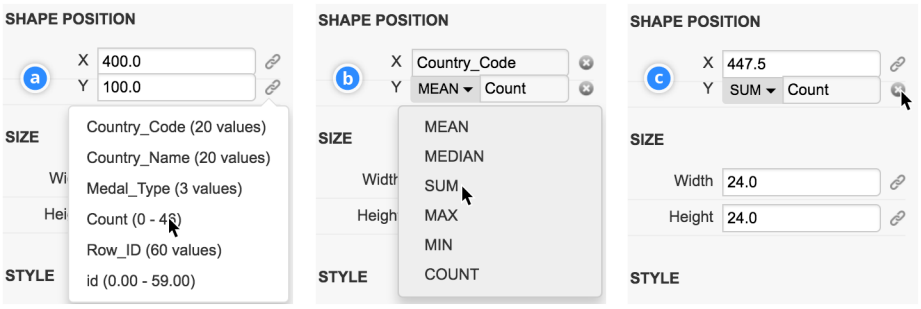


Figure 3.10: Lazy data bindings: (a) clicking the binding icon shows a list of applicable variables, (b) changing the aggregator when binding numerical variables, (c) property control and icon update after binding, the remove icon indicates the possibility of removing a binding

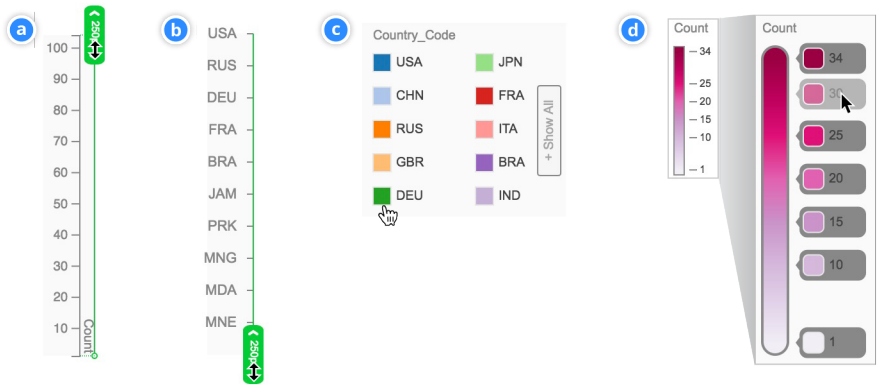


Figure 3.11: Configurable axes and legends: (a) Numerical axis. (b) Categorical axis. (c) Categorical color legend. (d) Numerical color legend.

approximate layout based on data values, and then remove those constraints to adjust their position for a desired free-form layout.

Interactive Axes and Legends – Data Illustrator automatically creates interactive axes and legends upon successful completion of data bindings. Axes and legends in Data Illustrator are explanatory - acting as a reference for the data binding, and configurable - supporting direct manipulation of the underlying scale. Upon their creation, Data Illustrator momentarily highlights axes and legends to help the user discover the afforded interactions. Users can adjust the offset of an axis with constraints: they can only move an x axis vertically, and a y axis horizontally. When the user moves an axis' peer objects, the axis follows, retaining its relative position. The user can click+drag on the axis handle to configure the underlying scale's range (Figures 3.11.a and 3.11.b), which updates the bound objects instantaneously. For categorical axis, users can define the order of objects in two ways: 1) sort the objects in a collection through the "ordered by" property control slider, 2) directly drag the axis text labels to reorder objects.

Color legends can be re-positioned anywhere on the canvas. They augment designs of color palettes and color gradients from mainstream vector editors. For categorical color legends, users change colors to replace the default colors picked by the system (Figure 3.11.c). For numerical color legends, users can select the color for each stop, add a stop

by a single click, and remove a stop by dragging it away (Figure 3.11.d). Changes to a color legend are immediately applied to its bound peers. Furthermore, axes and legends of the same semantic type can be linked or merged to support consistent data bindings across collections - a concept pioneered by Claessen and Wijk [133].

3.3.3 Software Development

The Data Illustrator system extends the Paper.js [134] scene graph and renders graphic primitives in HTML5 Canvas. Paper.js provides an object model for objects in a traditional vector graphic tool such as layers, groups, and vector paths. All user interactions in the system are handled with RxJS [135] – a functional reactive programming library that operationalizes input events as modifiable streams. All system-level actions are broadcasted using the Backbone.js [136] library. The system infers data types of each column with the Datalib [132] library and provides data summaries that are displayed in the *variables Panel*. All data bindings are implemented with the *d3-scale* module [10].

3.4 Results

3.4.1 Example Gallery

We have created a diverse set of visualizations using the Data Illustrator system to demonstrate the expressivity of the framework. The visualization examples and videos showing the authoring processes are available at http://www.data-illustrator.com/gallery.php. In terms of chart type, the examples include rectangular bar chart, triangle bar chart, grouped bar chart, stacked bar chart, scatter plot, composite scatter plot, bubble plot, line chart, multi-series line charts, slope chart, bump chart, heatmap, parallel coordinates plot, alluvial diagram, mosaic plot, box-whisker plot, range chart, waffle chart, Gantt chart, and stringline chart, and small multiples.

3.4.2 Usability Study

We focus on visualization re-creation tasks to evaluate whether designers can understand and use the framework to compose visualizations. While the re-creation task is not an exact replica of the complete design process, it allows us to choose visualizations that cover all the concepts and features in our tool, and to compare participants' performance objectively. Furthermore, the ability to think and act in terms of the framework concepts is the cornerstone of using Data Illustrator for ideation and more open-ended designs.

We recruited 13 designers (7 male, 6 female) from the Puget Sound area and the Atlanta metropolitan area. The breakdown of their experiences in graphic design is as follows: less than 2 years: 1 (7%); 2-4 years: 3 (23%); 4-6 years: 6 (46%); 6-8 years: 1 (7%); greater than 8 years: 2 (15%). Their design work included web UI (85%), mobile UI (77%), visualization and infographics (61%), graphics and illustration (54%), print (54%), logo (38%) and video game (7%). Out of the 13 participants, 5 (39%) had minimal or less than 2 years of experience with visualization, 4 (31%) had 2-4, 2 (15%) had 4-6, and 2 (15%) had more than 6 years of experience.

The study with each participant lasted 1.5 hours. In the setup we used two monitors, each with a resolution of 2500x1600. We first gave a tutorial on Data Illustrator following the script at https://goo.gl/UtZruK. The participants learned about the main features of the system by creating three simple visualizations: a stacked bar chart, a scatter plot, and a triangle bar chart with both positive and negative values. The tutorial lasted around 35 to 40 minutes. The participants then were asked to complete two visualization creation tasks. At the end of the two tasks, they could decide if they wanted to work on an optional, more difficult task, if time allowed. These three visualizations covered the main features and functionality of the system:

1. "Obesity vs. Education" – *required*
    - Repeating a line by data

- Binding numerical data to position of anchor points
- Binding categorical data to color of line and anchor points
- Merging scales

2. "NBA Redraft" – *required*
    - Repeating and/or partitioning rectangles by data
    - Interacting with nested collections
    - Binding numerical data to fill color
3. "Red and Blue America" – *optional*
    - Repeating and partitioning rectangles by data
    - Interacting with nested collections
    - Breaking a collection layout
    - Binding numerical data to fill color
    - Binding numerical data to position of segments
    - Binding numerical data to position of collections

For each visualization, we explained the schema and meaning of the source data, and described what the visual components and their properties represent. We asked the participants to focus on the main visualization and not to worry about the annotations. At the end of the session, each participant completed a questionnaire and answered questions in a semi-structured interview.

All participants successfully completed Tasks 1 and 2 with minimal guidance (μ=12.23 minutes, σ=5.61 for Task 1, μ=10.77 minutes, σ=4.30 for Task 2). Out of 13 participants, 12 volunteered to work on the third task. Four of them completed it successfully (μ=14.75 minutes, σ=2.87), the remaining eight could not finish the task after spending

time (μ=10.63 minutes, σ=5.93). For these eight participants, we analyzed how close they were from success. Completing Task 3 required four milestone steps to be accomplished:

1. Create nested collections with Year and State
2. Position the nested collections on a map layout
3. Bind Inclination to the fill color of the rectangles
4. Bind PVI to the y position of the rectangles' top segments

Seven out of the eight participants were able to finish two of the steps but were stuck on the final two steps. While the participants were not asked to complete the tasks as quickly as possible, their completion times were in the expected range for an efficient authoring tool. A designer could plausibly take 30 minutes to manually create each of these visualizations in a vector graphic editing tool (without data support). On the other hand, the results from a similar user study to evaluate Charticulator [27] reported faster average completion times (4 minutes) to re-create different visualizations from the ones used in our study (yet of comparable complexity). The difference between completion times could indicate that Charticulator is easier to use than Data Illustrator. However, a comparative study would be needed to directly compare the usability of these two systems based on the same pool or participants and tasks. Completion times could be attributed to differing study setups. For example, the moderators in the Charticulator study provided participants with hints whenever the participant stalled on a task, while our study design prevented moderators from giving participants help with the user interface.

The participants rated their experience of learning and using Data Illustrator on a 5-point Likert scale. The results are as follows: on learning, μ=2.62, σ=0.96 (1-very easy, 5-very difficult); on creating visualizations, μ=2.38, σ=0.77 (1-very easy, 5-very difficult); on the authoring experience, μ=2.15, σ=0.90 (1-very enjoyable, 5-very frustrating).

Designers' background and expertise directly affected their learning experience and performance. For those who had substantial experience with visualization, they thought

learning was easy: "*Tableau has a bit of a learning curve, and with Data Illustrator being based off of Adobe Illustrator, there isn't as much of a learning curve.*" (P3). In contrast, P5 had little experience with visualization, and compared to learning with graphic design tools: "*it takes 30 minutes for me to learn the [Data Illustrator] tutorial via a person, that usually to me is not an easy program. [Adobe] XD for me was easy 'cause I didn't have to use any tutorials, so I'd say [learning with Data Illustrator] is somewhat difficult*".

The participants were impressed by the power of the tool: "*Very impressive. When I looked at all 3 visualizations I thought: oh boy, how am I going to do this! Then once you finally work through the sequences needed to make it, the actually-doing-it part is super easy!*" (P11). P9 commented on the tool's flexibility and ease of use: "*I feel like it's more flexible than D^{3}or Tableau. It's a happy medium of being able to control the graphic visually. It's pretty simple too, you don't have to be a super expert user like with Adobe Illustrator, which is nice. It's a nice sweet spot between having little control with Tableau and getting frustrated with D^3*". P6, however, knew little about visualization and did not understand the concept of scale. He struggled in the authoring process, but still managed to complete the two tasks by trial and error.

We also observed that the designers exhibited different workflows in the authoring processes. In Task 1, some participants used the *Repeat Grid* to generate a few lines first, bound data to the anchor point positions, then used the *Peer Count* slider to generate the remaining lines; other participants generated all the lines first before binding data to positions. The strategies to accomplish Task 2 also varied. Some participants generated all the rectangles by Row ID, broke the grid, and bound data to the x and y positions; others saw a nested structure in the visualization, and repeated a rectangle by Year then partitioned the rectangles by Player. This diversity of workflow demonstrates the flexibility of our framework and system.

We identified three recurring pain points in using the system. First, many participants confused the order of shapes inside a collection with their positions. In Task 2, they wanted

to generate the visualization by simply sorting the shapes in a *Repeat Grid*. The order did determine the shapes' positions in a collection to a certain extent, which was the source of confusion. Second, in the current design, to bind data to shapes' position, one must break the layout. Otherwise the position property controls are hidden. Some participants were baffled not seeing the position controls. Showing the position controls at all times and prompting to break layout can resolve this problem. Finally, several participants could not recall the feature of binding to segment position and adjusting the scale to generate bar charts with negative values in Task 3. Binding to height felt more natural to them.

The participants also made suggestions on how to improve the interface and system. The lack of undo functionality bothered many participants. They were afraid of making irreversible mistakes and chose to think deeply about the authoring strategy before trying it out. They also wanted a more robust *Pen Tool* and the ability to style the axes and draw grid lines. Some participants also commented that starting from scratch was harder than picking a template: "*It took me some time [...] to think through sequences I would need to take to re-create it. Tableau has the 'Show Me' feature that hints what bindings can be made with that type of dataset*" (P3). Such comments are consistent with previous research findings [13]: compared to automated visualization generation tools, design-centric authoring tools evoke deeper reflections on design choices and execution plans. For use cases where quick visualization construction is desirable, saving the visualizations as reusable templates will be very useful.

3.5 Discussion

The Data Illustrator framework provides descriptive and generative power for visualization design, but in its current form, it is not complete. The framework needs to be expanded to include area as a shape primitive, so that it can describe visualizations such as area charts and stream graphs. Further research is also necessary to include the support for hierarchical, network and geographic data, and the corresponding visualizations. Also the framework

does not support layouts such as packing and polar coordinate positioning.

Data Illustrator also needs a more structured approach for handling relative positioning of data-bound graphic elements within a group. We also do not support a method for creating small multiple charts. A limitation of our approach is that we must implement and contrive how designers will use framework operations to achieve various data graphics. Each implementation of a framework operation includes the definition of appropriate heuristics. Instead we should consider an adaptive approach in the future such as machine learning techniques or constraint-based solvers.

Systems such as Lyra build on top of Vega, which have access to the functionalities offered by D^3, including interpolation methods for lines and curves. Data Illustrator should provide those capabilities in order to be more powerful. For future work, we would also like to explore how to turn visualization designs into reusable templates (further discussed in Section 5.4). Once users create a visualization inside Data Illustrator, they should be able to export it into formats readable by other tools and to share it with other users, who can customize the design with their own data and visual styles. Adding authoring support for interactivity is also a direction to investigate further (further discussed in Section 5.5).

CHAPTER 4

DATA ANIMATOR

In this Chapter, I introduce our approach for empowering graphic designers to author animated data graphics. Through our formative research, design, and implementation of the Data Animator system, I address research questions RQ1, RQ2, & RQ3 for animated visualizations. Data Animator is a continuation of Data Illustrator, graphic designers author animated transitions between static visualizations imported from Data Illustrator – called *vis boards*. Our formative research includes a survey of the design space and an ideation study with graphic designers. The outputs from our survey, animation primitives and compositions, informed the Data Animator framework by describing configurations for visualization states to change at the data, object, or graphic level. Data Animator builds upon the Data Illustrator framework to analyze and match objects between two static visualizations, and generates automated transitions by default (RQ1). Furthermore, results from our ideation study informed the design of Data Animator, as we found evidence for designers' preferences for keyframe animation and ideas for novel user interfaces and interactions when authoring data-driven animation. To address RQ2, Data Animator introduces a dual view design (Storyboard and Timeline) that augment familiar graphic design tools. In the Storyboard View, designers specify connections between vis boards to automatically generate an animated transition. Data Animator uses a matching algorithm to analyze the relationships between objects across vis boards. By default, animation is created by tweening the matched objects and applying animation effects (e.g., fade in, fade out) to the unmatched entering or exiting objects. This approach supports designers to quickly specify animation via automation. Data Animator supports the division of a complex animation into stages through hierarchical keyframes and uses data attributes to stagger the start time and vary the speed of animating objects through a novel timeline interface. To address

RQ3, we validate Data Animator' expressiveness via a gallery of examples, and evaluate its usability in a re-creation study with designers. The formative work in this Chapter is published as a conference paper [36] and the system is currently under review [37].

Contributors – The majority of the completed work in this chapter is attributed to me, as the first author. Zhicheng Liu of the University of Maryland (previously Adobe Research) contributed numerous design ideas, supervised my implementation efforts, and assisted in framing the contributions of this work. While Wilmot Li of Adobe Research and John Stasko of Georgia Tech provided advisory assistance and guidance on my research efforts.

4.1 Introduction

Animated data graphics are an increasingly popular medium for data-driven narratives on media outlets and digital publications such as *the New York Times* [137] and *the Pudding* [138]. These digital narratives present visualizations of data to help communicate information to the viewer. An increasing number include animated transitions between the visualizations. Although animation is not always beneficial for analytic tasks [139, 140], carefully designed animation of charts can facilitate detection of objects entering into a scene [141], enhance understanding and engagement through staged transitions [54, 55], and help track changes in data [57, 142].

Designing effective animated data graphics often requires thoughtful considerations on how to *coordinate the behavior of visual objects* and *pace the temporal rhythms*. For example, animated transitions can be localized in a specific component (e.g., an axis changes from linear to logarithmic scale), or happen between two completely different visual states (e.g., changing from a scatter plot to a stacked bar chart). In the former case, displacing the axis ticks and labels and updating the objects' positions might be sufficient; in the latter case with more drastic changes, designers need to decide which visual objects should enter

or exit the scene, and which objects need to be merged, transformed or split.

In addition to coordinating the behavior of visual objects, the temporal pacing of animation also requires deliberation. *Staging* and *staggering* are two commonly used techniques to pace animated transitions. Staging divides a complex animation into a sequence of simpler sub-transitions called stages. Staggering applies an incremental offset to the starting time of each moving object, thus avoiding the confusion caused by all the objects moving simultaneously. These techniques may be manually designed but are often driven by data. For example, staging may be performed based on the data hierarchy, while the parents animate first, and the children later [142]; the trigger time and the speed at which each shape animate can be specified as a function of data values in an attribute.

Existing animation authoring tools, however, lack support for coordinating visual objects and specifying data-driven temporal pacing. Prevalent authoring paradigms for animated graphics include:

- *keyframe animation*: specify properties of graphical objects at certain points of time by setting a set of keyframes, frames in between two keyframes are generated by tweening.
- *procedural animation*: generate animation of large number of animated objects with a set of behavior parameters.
- *presets & templates*: apply predefined animation effects and configurations to objects.

In Section 4.3 we investigate designers' preferences for these three authoring paradigms when data becomes an integral part of animation design. Moreover, we analyze 52 examples of animated data graphics collected in the wild. The analysis of these real-world examples yields a design space, expressed as primitives in four dimensions: *object*, *graphics*, *data*, and *timing*. These primitives provide building blocks for higher-level compositions

such as transition types and pacing techniques. Under the backdrop of the animation authoring paradigms and the design space, we conduct an ideation study to understand how designers conceptually approach authoring animated data graphics.

The study results show that keyframe animation is a familiar authoring paradigm preferred by many participants in a majority of tasks. However, we find evidence that an authoring tool should combine paradigms as the other two paradigms are indispensable for certain tasks. Under the keyframe animation paradigm, to perform visual object coordination, it is necessary to track and manipulate objects across different visual states or frames. It is easy to do so in traditional keyframing tools such as Adobe After Effects [29] for animated graphics, where the number of objects is typically small. Animated data graphics, however, often contain hundreds or more visual objects, so analyzing and tracking the objects become a challenge. Selectively applying different behaviors to different groups of objects for coordination is also non-trivial. In order to specify and control temporal rhythms, designers need to articulate how data attributes may drive the staggering of objects or map to the speed of transition. No existing keyframing tools support these tasks with ease and precision. A few template-based tools provide predetermined temporal designs, but the range of expressivity is limited.

To date, programming remains the only viable way to create expressive animated data graphics. For example, D^3 [10] leverages the Document Object Model (DOM) for interaction and graphics rendering, and provides the enter, update, exit paradigm to manage the differences between current and next visual state of a data graphic. D^3 also provides a transition management module for interpolating any objects' properties *en masse*, while supporting coordination of temporal aspects of animation (e.g., delay, duration, easing). However, programming takes significant time to learn, write, and preview designs. Designers often favor familiar design tools for animation and prototyping that provide expressivity and rapid feedback.

We introduce Data Animator, a system for authoring animated data graphics without

programming. Our approach is rooted in the keyframe authoring paradigm, and incorporates elements from the preset & templates paradigm when necessary. Animated transitions are specified between static data graphics – called *vis boards*. Data Animator adopts a dual view design (Storyboard & Timeline views) that lets users switch between levels of authoring granularity.

In order to support visual object coordination and data-driven temporal pacing, the system needs to understand the structure and properties of static visualizations. To this end, we build Data Animator on top of the Data Illustrator framework, which describes the visual properties, data bindings and organizational structure of objects in a static visualization.

To facilitate coordinating visual objects, Data Animator uses a matching algorithm to analyze the relationships between objects across vis boards. Animation is created by tweening the matched objects and applying animation effects (e.g., fade in, fade out) to the unmatched entering or exiting objects. This approach supports rapid generation of animations through automation. Since automated matching may fail in certain cases, Data Animator provides a novel user interface for designers to visualize and interpret the results of the matching algorithm, and to override defaults and manually adjust transitions.

To support the authoring of data-driven temporal pacing, Data Animator supports staggering by data where the value of a data attribute determines the delay of the animating objects. We also introduce a concept called hierarchical keyframes, where the allocated duration for a transition cascades as a linear function of the parents' duration. Hierarchical keyframes allow the creation of expressive pacing by combining staging, staggering and grouping of graphical objects. We present multiple scenarios to illustrate this concept and discuss the design of a novel interface for specifying and visualizing hierarchical keyframes.

Finally, we demonstrate the Data Animator system's expressive capabilities via a gallery of examples and evaluate its usability from a re-creation study with 8 designers. All the participants could complete each of the six animation re-creation tasks within a few min-

utes. Users found Data Animator useful for supporting the daunting task of coordinating numerous objects and provided feedback on how to improve the system interface further.

4.2 Survey of Animated Data Graphics

Animated data graphics are composed of *animated transitions* on data graphic elements. An *animated transition* is the interpolated change between two *visual states*. We further breakdown these terms as follows:

- A *visual state* can be thought of as a data graphic at rest. Visual states include *all* information about the data graphic's static design.
- A *transition* is the uninterrupted change from one visual state of a data graphic to another.
- *Animation* is the sequence of intermediary states between a transition that is typically perceived as continuous movement.

4.2.1 Methodology

To understand the design space of animated data graphics we surveyed examples from online sources. The purpose of our survey is to *(i)* identify composable primitives of animated data graphics, and *(ii)* recognize common compositions of those primitives. Our survey includes data narrative articles, data videos, visualization slideshows, and interactive maps. We collected an initial set of 78 examples from the following sources: the Kantar Information is Beautiful Awards [3]; media outlets including the New York Times [137], the Pudding [138], the Google News Lab [101]; and well-known freelance designers such as Nadieh Bremer [143], Neil Halloran [144], Shirley Wu [145], and Moritz Stefaner [146].

We exclude examples where all animated transitions are unrelated to data. For example, we exclude user interface transitions such as highlighting buttons or expanding menus. We also exclude animated transitions that only occur at the frame level – meaning none of

the constituent objects transition. By only including examples with data-related animated transitions, we refined our initial pool to the final collection of 52 examples. This survey is not exhaustive; however, our goal is not to catalogue all animated data graphics but to inform the design of authoring tools. To that end, our survey includes a diverse set of data graphic forms and animated transition designs.

To analyze these examples, we first identified their unique animated transition instances. We identified instances based on the following criteria: *(i)* an object undergoes an animated transition, *(ii)* the animated transition is data-related, *(iii)* repetitive animated transitions count as a single instance (e.g., circles in a scatterplot change y-position). When analyzing these animation instances, we set out to identify building blocks for authoring tools that go beyond templates. Related taxonomies [54, 33] provide transition types that are well-suited for animation templates. While templates are easy to use and understand, they are difficult to customize. We seek to identify composable primitives for authoring tools that are generalizable across datasets and visualization forms.

During weekly meetings over a period of three months, we iterated on these primitives until we could accurately describe each animated transition instance. During this iterative process we sought to balance the granularity of our primitives. Too high-level and the primitives would resemble animation templates, while low-level primitives would not be generalizable across datasets and visualization forms. We identified composable primitives from each of four dimensions: *object*, *graphic*, *data*, and *timing*. The *object* dimension refers to *what* type of graphic object transitions during the animated transition. The *graphic* and *data* dimensions describe *how* the object transitions from one *visual state* to the next. Finally, the *timing* dimension consists of relative timing concepts to compose animation designs and pace sequences of animated transitions.

4.2.2 Primitives

Here we provide a description of the design space for animated data graphics. We characterize the design space by identifying composable primitives across four dimensions: *object*, *graphic*, *data* and *timing*. The primitives combine to form animated transition designs.

Dimension 1: Object Type

The *object* dimension describes *what* type of graphic object undergoes a transition. We propose the following set of 5 object types, delineated by their *role* in a data graphic and their unique properties. The same graphical element can be used in a variety of ways, differing by its role, relation to data, constituent parts and attributes. For example, a rectangle can be used as a glyph to represent data and encode data values using its visual attributes; a rectangle may also be used as a legend or an annotation.

Glyphs are graphical marks (e.g., lines, rectangles, text, images, groups) representing one or more data tuples. A glyph's visual appearance may encode data values.

Groups are collections of glyphs. Glyphs within a group are often arranged in a spatial layout.

Axes & Legends are the visual representation of scales that map data values to visual properties. They explain how data maps to visual space of the data graphic.

Annotations are auxiliary elements that help explain key insights and contextual information about the data graphic to the audience. Annotations contain text, shape, and image elements and target different components of the data graphic (e.g., specific glyph, series of glyphs, visual substrate, entire graphic). Annotations include labels, captions, tooltips, and footnotes. Annotations are not bound to data, although they may depend on data attribute-values from a target glyph.

Cameras provide a configurable vantage point of the data graphic. The camera (or viewport) projects the scene graph onto a view based on the camera's configuration. The projection attributes depend on camera type, but can include focal point, field of view, zoom

level, and rotation. Changes to these parameters result in pan, zoom, and rotation actions.

Dimension 2: Graphics

The *graphics* dimension describes *how* an object changes from one visual state to the next. It is concerned with the constituent visual elements, their visual attributes, and configurations that compose a data graphic scene. Initially, we considered describing the low-level visual properties that change for each object (e.g., visibility, position, opacity, fill color, text content). However, this approach only provides a taxonomy of the visual attributes that transition in our set of examples. Instead, we provide 3 primitives that describe how an object visually changes.

Visual Presence is the existence of an object. When an object is added or becomes visible, we say it "enters" the scene graph. Conversely when an object is removed, we say it "exits."

Visual Attributes are the visual channels of objects such as position, fill color, stroke, opacity, etc. that are set by the designer and unrelated to data encodings. Unless overridden by data encodings or configurations, glyph instances share the same visual attribute values. Visual attributes differ for graphical elements (e.g., text elements have different visual attributes than line elements).

Configurations are the parameters of an object that are not directly visible in objects. Configurations differ from visual attributes because the parameter value is not directly expressed as a visual value. For example, a group can have a layout configuration, specifying how its constituent glyphs should be positioned.

Dimension 3: Data

The *data* dimension describes *how* the underlying data changes in a data graphic. Modifications to data mappings can change visual properties of objects. For example, if the bound data attribute is changed in a mapping, the glyphs may express a new visual value for the newly bound data values. Change in the underlying data at times results in visual changes

to glyphs, however this differs from the *graphic* dimension where each glyph expresses the same visual attributes regardless of data.

Data Presence is the existence of data tuples. Data is added or removed to a data graphic through manifestations in objects. For example, glyph instances are bound to one or more data tuples.

Data Encodings are the functions that transform data attribute values into visual property values for each glyph instance. Data encodings can be shared across groups of glyphs.

Data Transforms are the operators that manipulate datasets into new forms to then be attached to graphical objects. Data transforms include data nesting, aggregation, and linking. These operations allow for the specification of data graphic designs beyond the raw dataset.

Data Targets are the data focal points for other objects. For example, annotations such as labels and tooltips target a glyph for a specific data tuple.

Data Queries are predicates applied to a dataset that generate inclusion and exclusion selections. The design specifies how the corresponding glyphs for these two selections will be visually altered. For example, filtering temporally removes the exclusion selection from the scene graph, while highlighting visually emphasizes the inclusion glyphs and/or de-emphasizes the exclusion glyphs.

Dimension 4: Timing

The *timing* of an animation describes the pace at which visual properties successively move from start to end of a transition. Although a transition has the same start and end visual states, animation provides diverse opportunities to illustrate between those states. We describe timing of animations based on 4 primitives. The primitives rely on a relative notion of timing. For example, the *duration* of an animation is defined as the relative amount of time that transpires between the start and end of an animation. Relative timing allows designers to compose successive animations into larger animation compositions.

Triggers are events that initiate an animation. Triggers provide an initial reference point.

Triggers include the following: a specific timestamp indicating the start or end of another animation, repetitive temporal events such as a clock, or navigation inputs such as scroll, button clicks, or slider events [51].

Delay is the time to start a transition relative to the trigger point. Zero delay coincides with an immediate start of the animation.

Duration is the amount of time to complete a transition. Duration is defined in relation to the animation's start point as the amount of time that transpires until the animation ends.

Easing specifies the speed that a transition progresses at different points in time of the animation. An easing function computes the value of an animated property based on the percentage of time that has progressed in relation to the duration.

4.2.3 Compositions

The dimension primitives discussed in Section 4.2.2 are intended to be combined into compositions of animated transitions. In this section we identify commonly employed compositions from our survey of examples. We recognize a set of 10 transition types based on the primitives from the *object*, *graphic* and *data* dimensions. We also point out popular pacing techniques composed from the *timing* primitives. These compositions are not a taxonomy – they do not describe all possible animated transitions. However, these compositions can be combined further to create more complex transitions and animations.

Transition Types (object, graphic, data)

In our design space, transitions are described by the changes in the *object*, *graphic* and *data* dimensions. When a transition is not specific to an object primitive, it can be applied to multiple object types. We identified a set of 10 commonly employed transition designs found in our survey of examples. It is obvious that these transition types are not exhaustive: there are many more possible primitive compositions.

Enter/Exit: (visual presence + data presence) A data-bound object is entirely added or

removed from the scene graph. The change occurs in the object's visual presence and data presence. Applicable object types include glyphs, objects, annotations, and axes and legends. An example of enter/exit is the introduction of a new glyph or chart.

Combine/Partition: (data transform + visual presence + configuration) Objects are combined or partitioned based on a nesting data transform and re-arranged by a change in layout configuration. Each objects' visual presence changes. On aggregate, the attached data tuples are not added or removed from the collection (i.e., data presence), however data tuples are re-attached to objects via data transformation. An example is the partitioning of a bar chart using a data attribute, resulting in a stacked bar chart.

Visual Alteration: (visual attributes) Here an object's visual attributes change, which is specified by the designer and independent from data. The object's data encodings, data transforms, data presence remain unchanged.

Data Encoding Alteration: (data encodings) Data encodings are altered, added, or removed from the data graphic. This change may or may not result in changes in terms of visual presence or visual attribute. For example, if the data value remains unchanged, so does the visual attribute.

Ordering/Re-configuring: (configurations) This transition typically applies to a group, where its layout configuration (graphical dimension) changes, resulting changes in the spatial positions (graphical dimension) of its constituent members. This configuration change may be based on data (e.g., sorting by a data attribute), or may be based on stylistic considerations only.

Highlighting/Filtering: (data queries + visual presence — visual attributes) Objects are visually highlighted or filtered based on the inclusion or exclusion selections defined by a data query. Here the relevant graphical dimension primitives include visual presence and visual attributes, and the relevant data dimension primitive is data queries. Applicable objects include glyphs, objects and annotations.

Data Ticker: (data presence) The visual states cycle through the values of a temporal or

nominal data attribute. This attribute is typically orthogonal to the data attributes represented in the graphic. This type of transition applies to glyphs and objects. An example of data ticker is the animated bubble chart over time.

Appear/Disappear: (visual presence) Objects visibility changes in the scene. The main difference between this type and enter/exit is that enter/exit involves data presence as a primitive but appear/disappear does not. Examples of appear/disappear include the introduction of an annotation, which is not bound to data.

Camera Alteration: (camera + configurations) The camera's configuration such as position or projection properties are altered, resulting in a view change (e.g., panning, zooming, rotating).

Simulated Process: (data transforms) Glyphs animate based on a simulated process, defined by an underlying algorithm or streaming data source.

Pacing Techniques (timing)

Here we describe popular pacing techniques based on the *timing* primitives discussed in Section 4.2.2. The following compositions rely on relative timing concepts to build up successive animations into larger compositions. Pacing techniques are designed to assist the audience perceive and apprehend how data graphics change during animated transitions.

Staging segments an object's visually complex transition into sub-transitions, allowing for multiple changes to be easily observed. Staged transitions rely on subsequent sub-transitions to trigger after the previous sub-transition ends. Heer and Robertson [54] demonstrated that staged transitions are preferred for visually tracking changes and slightly reduce errors.

Layering is similar to *staging*, as sub-transitions precede after one another. However, layering typically is applied to introduce objects of a data graphic in a piece-meal fashion. Also, momentary pauses or *dwells* are applied between the sub-transitions. Layering follows a formula of sub-transitions triggering after the previous sub-transition ends, with a

delay in-between called a *dwell* that allows the audience to apprehend newly introduced objects. According to Schwabish [147], layering is an effective technique to manage the audience's attention as you ask them to follow the progression of presented information.

Staggering is the incremental or distributed delay of animations' start times across a collection of objects' sub-transitions. Typically staggering is applied to data glyphs or axes & legends as the ordering of start times is based on data or visual attributes. The mapping for each objects' animation start time can be defined by sequential, linear, ordinal, or cardinal functions from a single trigger point. Staggering only relates to the start time of an objects' sub-transition; sub-transitions are not required to precede each other. Our formulation of staggering differs from that of Chevalier et al. [56], where objects move after the previous object comes to rest. They found that staggering has negligible, or even negative, impact on an observer's ability to track multiple objects.

Looping is a cyclic succession of transitions that loops back to the start. Sub-transitions occur one after another in a cycle. Loops are similar in form to *staging and layering*, however the initial sub-transition animates after the last transition - thus closing the loop. This repetitive technique is often used to display cyclic temporal data. According to Lena Groeger of ProPublica: "looping makes us notice differences because our attention can shift around to different places" [148].

4.3 Semi-Structured Ideation Study

To understand how designers think about and approach authoring animated data graphics, we conducted an ideation study with participants possessing experience in graphic, visualization, and animation design. We use the three paradigms from Section 2.4.3 as inspirational exemplars and asked participants to discuss how they would author six given animated transitions *conceptually*. We seek to answer the following three research questions:

- Q1: Among the graphical, temporal, and data components of an animated transition,

which are more commonly associated with a preferred authoring paradigm?

- Q2: How do the characteristics of an animated transition (e.g., transition type) affect the participants' choice of authoring paradigms?
- Q3: What implications do the participants' preferences have on the design of authoring systems and interfaces?

We instructed the participants to draw from their previous experience designing animated graphics, but also to conceive of computer-assisted concepts that would be useful for manipulating data-bound visualizations and transitions. We chose this semi-structured ideation format over a re-construction task with an existing tool (e.g., Adobe After Effects [29]) because we did not want a particular tool to constrain the participants' thinking. We wanted designers to consider all relevant authoring paradigms. Furthermore, creating animated data graphics with current design tools is time-consuming; an ideation study allows participants to experience a breadth of animated transitions in a reasonable amount of time.

4.3.1 Tasks

For each task, we asked participants to discuss how they would author a given animated transition. We selected 12 animated transition instances from our survey, these instances originate from the following 4 examples: *The Timing of Baby Making* by Amber Thomas [149]; *Twenty Years of the NBA Redrafted* by Russell Goldenberg [122]; *I'm Not Feeling Well* by Gabriel Gianordoli [150]; *A visual introduction to machine learning* by Stephanie Yee and Tony Chu [151]. We chose these animated transition instances based on their coverage of the design space from Section 4.2 and breadth of graphical complexity. The 12 instances represent the 10 transition types from our design space. We asked each participant to complete 6 tasks (6 of 12 total).

4.3.2 Participants

We recruited 14 designers (10 female, 4 male) from the Atlanta metropolitan area. On average the participants were 27.9 years old (min = 22; max = 39). Most of them are academics (2 undergraduate students, 10 graduate students), and 2 participants are working professionals. Their years of experience in graphic design is: less than 1 yr = 1 (7.1%); 1-2 yrs = 4 (28.6%); 2-5 yrs = 6 (42.9%); 5-8 yrs = 0 (0%); more than 8 yrs = 3 (21.4%). The distribution of their experience in creating charts, infographics, and data visualizations is: none = 1 (7.1%); less than 1 yr = 2 (14.3%); 1-2 yrs = 7 (50.0%); 2-5 yrs = 2 (14.3%); 5-8 yrs = 0 (0%); more than 8 yrs = 2 (14.3%). Participants reported using the following categories of tools to create data graphics: vector editors = 11 (78.6%); presentation tools = 11 (78.6%); spreadsheets = 8 (57.1%); visualization software = 7 (50.0%); programming toolkits = 7 (50.0%); image editing tools = 6 (42.9%); infographic tools = 1 (7.1%).

The prevalence of participants' experience in creating professional animations is: none = 0 (0.0%); less than 1 yr = 10 (71.4%); 1-2 yrs = 4 (28.6%). When asked about the infrequence of their experience creating animated graphics, many participants responded in regard to their professional experience creating animations, but they had additional years of experience in the classroom. Participants reported creating the following categories of animations: UX animation = 8 (57.1%); interactive visualizations = 6 (42.9%); effects animation = 4 (28.7%); 3D animation = 4 (28.6%); character animation = 3 (21.4%); motion graphics = 3(21.4%); data narratives = 2 (14.3%); stop motion = 0 (0.0%). Participants reported using the following tools for creating animations: Adobe After Effects = 9 (64.3%); InVision = 6 (42.9%); Microsoft PowerPoint / Keynote = 5 (35.7%); Blender = 3 (21.4%); Autodesk Maya = 2 (14.3%); Programming Toolkits = 2 (14.3%).

4.3.3 Experimental Setup

The study took place in a laboratory setting, with each session taking 1.5 hours. Sessions were audio recorded and participants interacted with a 2880x1800 screen laptop. Partici-

pants first watched an 8-minute informational video that explains fundamental data visualization concepts and the three authoring paradigms. The informational video highlights the differences between the three paradigms: keyframing as declaring a time and property for the system to tween between states; procedural as a system of rules that updates graphics based on specified parameters or events for each frame of the animation; and presets & templates as predetermined transitions that are applied to graphics using relative timing.

During the task portion, participants worked on 6 (out of 12) total animated transition instances originating from 2 (out of 4) examples. For each task, participants first familiarized themselves with the example on a Chrome web browser for 3-5 minutes to understand the context of the animations. Participants then completed a *worksheet-guided analysis* and an *authoring ideation task*.

Worksheet-Guided Analysis: We presented each animation as a slowed-down screen-capture via QuickTime Player. Upon viewing the animation, we prompted the participant to fill-out a worksheet. The worksheet assists in building and externalizing participants' understanding of the animated transition in terms of the *object*, *graphics*, *data*, and *timing* dimensions proposed in Section 4.2. The purpose of the worksheet was not to test participants' abilities to analyze animated transitions. Therefore, participants were provided answers upfront (during pilot studies, data change and duration were difficult to discern from a screen-capture alone) and corrected if they made an error during analysis. Completing the worksheet was a prerequisite for the participant to discuss how they would *conceptually* author the animated transition.

Authoring Ideation Task: We asked the participants how they would approach authoring the animated transition instance in a visual authoring system rather than textual programming. We prompted them to consider three authoring paradigms presented in the informational video and encouraged them to think creatively beyond the paradigms if necessary. We also asked the participants to think-aloud; drawing on past experiences with familiar tools to imagine useful system concepts. We encouraged participants to sketch ideas on paper.

Based on their initial ideas and sketches, we asked follow-up questions and urged participants to consider deeply how they would author each component of the animated data graphic.

Upon completing the 6 tasks participants answered debrief questions to summarize their ideas, provide additional thoughts, and reflect on their past design experiences.

4.3.4 Analysis

We analyzed the audio recordings from each task. We did not include the worksheet-guided analysis recordings, as they do not include discussions on authoring. In total, we recorded 84 completed tasks (6 tasks $\times$ 14 participants). We transcribed each task, including in-situ references to participants' sketches.

We sought to answer our three research questions by coding the transcripts as follows: First, code the authoring paradigm expressed by the participant (procedural animation, keyframe animation, and/or presets & templates); *Q1* - code the animated data graphic by the *object*, *graphic*, *data* and *timing* dimensions from Section 4.2; *Q2* - code each animated transition instance based on the transition types from Section 4.2; *Q3* - apply open coding to identify user interface ideas and system features proposed by the participant.

We only coded responses relevant to authoring. Many utterances were not relevant: participants often asked questions about the data graphic or the task at hand; some utterances were unclear on authoring intent; while others were part of the participant's attempt to formulate their thoughts. Two authors individually coded transcripts from 4 sessions (conditions A & B). We measured the inter-coder agreement on authoring paradigm codes using Cohen's Kappa (K) [152]. Based on Kandis and Loch's scale [153] our codes are in "substantial agreement" as K = 0.69. Once in agreement of codes, the primary author completed the coding for all 14 sessions.

Table 4.1: Distribution of 12 animated transition instances employed across conditions A & B. Results show percentage of participants that described an authoring paradigm for each task.

TASK ID	EXAMPLE NAME	TRANSITION TYPE	Keyframe KF	Procedural PD	Presets & Templates PT
CONDITION A					
A1	NBA Redraft	Ordering/Re-configuring	85.7%	28.6%	42.9%
A2	NBA Redraft	Highlighting/Filtering	85.7%	0%	85.7%
A3	NBA Redraft	Data Encoding Alteration	71.4%	42.9%	57.1%
A4	Baby Making	Appear/Disappear	14.3%	0%	100%
A5	Baby Making	Data Ticker	71.4%	42.9%	57.1%
A6	Baby Making	Combine/Partition	100%	0%	42.9%
CONDITION B					
B1	Not Feeling Well	Data Encoding Alteration	71.4%	14.3%	28.6%
B2	Not Feeling Well	Camera Alteration	57.1%	71.4%	28.6%
B3	Visual Intro to ML	Enter/Exit	85.7%	57.1%	42.9%
B4	Visual Intro to ML	Data Encoding Alteration	100%	71.4%	28.6%
B5	Visual Intro to ML	Visual Alteration	71.4%	71.4%	28.6%
B6	Visual Intro to ML	Simulated Process	42.9%	100%	0%
ALL TASKS					
			71.4%	41.7%	45.2%

4.3.5 Results

To help explain the results of our analysis, we denote the three authoring paradigms with the following shorthand: [keyframe (KF), procedural (PD), presets & templates (PT)]. Overall, across all authoring tasks (84 total, 6 tasks $\times$ 14 participants), the participants described authoring with keyframe animation most frequently [KF: 71.4%], followed by presets & templates [PT: 45.2%], and finally procedural animation [PD: 41.7%].

Q1: Which animation components are more commonly associated with a preferred authoring paradigm?

Co-occurrence analysis of animation components (e.g., object, timing, data) identified the elements that participants frequently modify when authoring with either of the three authoring paradigms. In this portion of the analysis, percentages are based on a denominator

equal to the total number of tasks discussed by participants for each authoring paradigm (e.g., 49 is the total number of tasks where participants discussed glyphs and keyframe animation; 60 is the total number of tasks where participants discussed keyframe animation; therefore during 49/60 or 81.6% of tasks, participants discussed authoring with keyframe animation using glyphs).

Object Dimension:

Data Glyphs: Participants most frequently employed procedural animation and keyframe animation when working with glyphs [**KF: 88.6%**, **PD: 81.7%**, PT: 23.8%]. Participants described procedural conditions or rules that would apply for all glyphs. Similarly, they also described keyframes as being repeated for glyphs. However, some participants struggled to come up with the "apply to all glyphs" concept – P7 claimed: "*If I don't have so many objects - maybe only 10 objects are moving. I think I'm comfortable with After Effects, but like this one, I guess there are hundreds of objects moving at the same and it must be complicated*." This is a fair comment if the participant must manually create hundreds of keyframes or procedures for hundreds of glyphs. However, the generative concept of "apply to all glyphs" or "repeat for glyphs" that many participants described alleviates this manual burden.

Visual States: Participants also considered the entire static graphic at rest – they expressed how they would create animations from visual states more frequently with keyframe animation and presets & templates [**KF: 48.3%**, PD: 20%, **PT: 42.1%**]. Participants described how keyframe animation works well for setting two different data graphics as frames and allowing the system to link the glyphs by data, interpolating the properties that transition between each state. Participants also frequently described applying presets & templates to frames in a "slide show" or "page viewer" interface. Participants described how presets & templates could be applied to an entire static frame to create a transition.

Groups: Furthermore, participants described groups as a way to structure animations, most frequently they described groups when working with keyframe animation or procedural

animation [**KF: 16.7%**, **PD: 14.3%**, PT: 5.3%]. Groups were typically created via procedural statements such as ”group glyphs by data attribute”. Participants desired to apply keyframes to groups of data glyphs to achieve staging – for example P2 explained how they would first “*create different groups for both blue and green [rectangles]*” to rotate them all together and then transition the rectangles y-position. Participants also desired groups as a means to order an otherwise cluttered timeline interface.

Timing Dimension: We considered how participants authored timing effects (e.g., delay, duration, easing) in relation to the three authoring paradigms. We found that participants often expressed the need to create functions based on data or visual attributes that specify delay functions for glyphs [KF: 43.3%, **PD: 68.6%**, PT: 44.7%]. We also found that participants described presets & templates as a method to stagger delay based on data or visual attributes, P4 referred to each glyph animating after the other glyph as the “domino effect.” Some participants described the process of creating staging frequently with keyframe animation and presets & templates [**KF: 11.7%**, PD: 8.6%, **PT: 18.4%**]. Participants described adding keyframes for visual attributes or encodings of glyphs to stage a complex transition; while other participants described the relational timing of presets & templates (e.g., “animate after”, “animate with”) as a straight-forward method to stage transitions after one another. When it comes to the duration of transitions, participants did not have a clear preference for authoring paradigms [KF: 30%, PD: 28.6%, PT: 34.2%]. Finally, participants seldom described easing functions [KF: 5%, PD: 2.9%, PT: 2.6%]. This could be related to a lack of identifying easing in the worksheet-guided analysis.

Data Dimension: We associated how participants described different aspects of datasets (e.g., data tuples, data attributes, data transforms) with each of the authoring paradigms. We found that participants most frequently described procedural statements that altered the data tuples bound to glyphs [KF: 23.3%, **PD: 34.3%**, PT: 10.5%]. Participants frequently described procedural statements to alter the data transforms of the glyphs to achieve a change in bound data tuples. Therefore, data transforms were more frequently associated

with procedural animation than other authoring paradigms [KF: 13.3%, **PD: 28.6%**, PT: 10.5%]. Participants most frequently used data attributes when describing procedural rules or statements [KF: 50%, **PD: 71.4%**, PT: 44.7%]. These procedural rules include the previously mentioned delay functions, pivoting data transforms by a data attribute, or creating rules for data encodings. Data attributes were also expressed as ingredients for the other two authoring paradigms. Participants created keyframes by altering a data encoding's data attribute. Participants also discussed creating pre-defined timing effects based on a selected data attribute.

Q2: How do the characteristics of an animated transition (e.g., transition type) affect the participants' choice of authoring paradigms?

The results show that participants have preferred authoring paradigms for certain transition types proposed in Section 4.2. In Table 4.1 we compare authoring paradigms across each task (row). The columns of Table 4.1 describe transition types and distribution of authoring paradigms expressed by participants. The rows do not total to 100% as participants expressed multiple authoring paradigms for a single task. In this portion of the analysis, percentages are based on a denominator of 7 participants completing each task.

Participants discussed creating appear/disappear transitions using presets & templates more than any other paradigm [KF: 14.3%, PD: 0%, **PT: 100%**]. Participants cited the "lack of data bindings" and "small number of graphic objects" as justification for using presets & templates. They also remarked that such transitions are similar to the appear/disappear presets from familiar tools. Participants described creating simulated processes with procedural authoring most frequently [KF: 42.9%, **PD: 100%**, PT: 0.0%]. Participants claimed that state-based simulations would most easily translate to procedural rules. According to participants, keyframe animation best suits combine/partition transitions [**KF: 100%**, PD: 0%, PT: 42.9%]. Keyframe animation provides a visual timeline to control the combination of objects between these key states. Participants also imagined that keyframing could link partial objects to combined objects based on underlying data

(and vice versa). Participants favored using keyframe animation or presets & templates for highlighting/filtering transitions [**KF: 85.7%**, PD: 0%, **PT: 85.7%**].

For all other transition types, participants did not have an overwhelming preference for an authoring paradigm. Zero participants described using an authoring paradigm for the following transition types: procedural animation for highlighting/filtering, appear/disappear, and combine/partition [PD: 0%]; and presets & templates for simulated processes [PT: 0%]. Among those remaining transition types, participants described authoring at least once for the three paradigms.

Q3: What implications do the participants' preferences have on user interface design and system features?

We asked participants to comment on useful user interface designs and system concepts for an authoring system during each task. Participants often explained how interfaces from familiar tools or completely novel concepts would be useful for authoring animated data graphics. In this section we analyzed the frequency of user interface designs and system concepts described by the 14 participants. Percentages are based on a denominator of 14 (14 total participants).

Storyboards: 9 of 14 participants [64.3%] commented on how a slide show or storyboarding interface would be useful for authoring animated data graphics. These storyboards serve as representations of high-level story points for quick specification, sharing, and preview. As seen at the top of P12's sketch in Figure 4.1.1 – the stages of an animation are displayed as storyboards with arrows between each storyboard that represent the animated transition between each stage. P12 envisions that "transition arrows" are selected to edit animation properties of that transition. Beyond smaller staged transitions, participants desired to create and layout slides or storyboards for each crucial story point in their larger narrative. P4 and P10 cited the need to quickly iterate on these key story points at a high-level and share storyboard ideas with colleagues or stakeholders before investing time to author animations between story points. 10 participants [71.4%] explained that animation

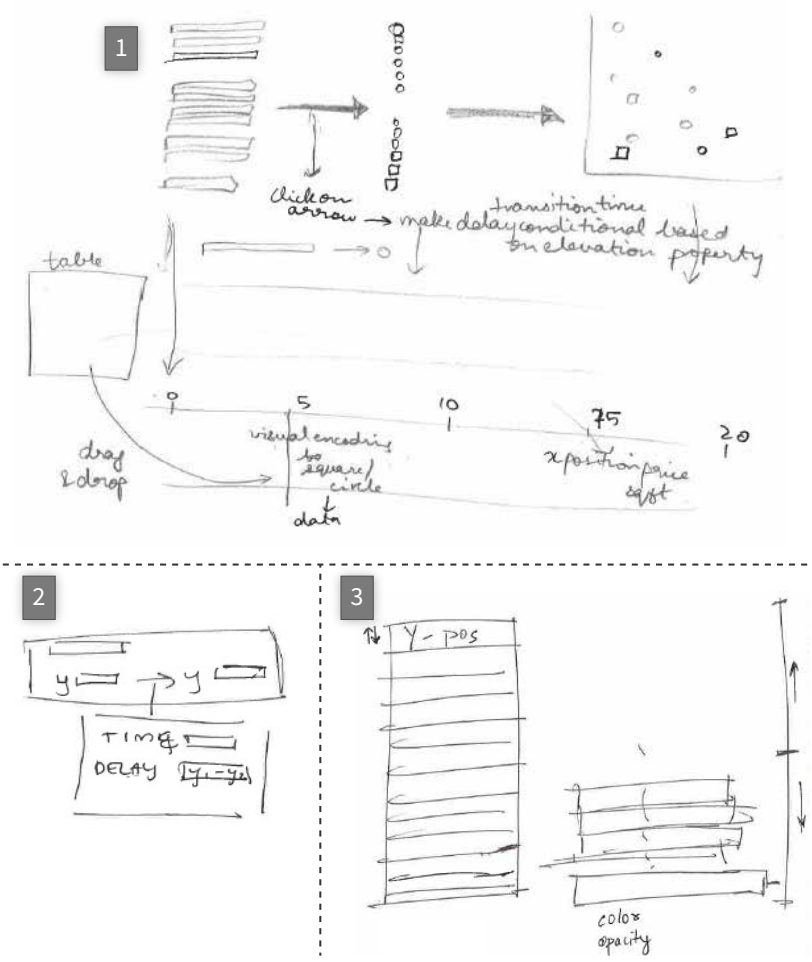


Figure 4.1: Participants' user interface sketches for an animated data graphic authoring tool. (1) P12's interface combines storyboarding and timeline views, (2) P5's concept for modifying a transition's time components such as delay functions, (3) P10's data table supports ordering the delay of glyphs' transitions by visual attributes such as y-position

previews would be helpful to compare and contrast potential animation design options. According to P14: "*Previews would help. When you set an effect...you can see the preview in the background...So that would be helpful because [the system] could create a GIF. You could watch it happen [and] decide if you need to modify [the animation] in real time.*"

Timelines: 8 of 14 participants [57.1%] described how they would want to use a timeline interface to author animations. This correlates with the participants favoring keyframe animation for the majority of tasks. As seen at the bottom of P12's sketch in Figure 4.1.1 – the stages of a transition could be created as keyframes on a timeline. Figure 4.1.1 depicts

P12's combined storyboard and timeline interface design. In this design, storyboards are linked together by transitions and further specification of transitions occurs in a timeline interface – this dual interfaces appears in related prototyping tools such as InVision [31]. We hypothesize that that a similar dual interface would be useful for authoring animated data graphics. Participants had two approaches for creating keyframes: by transforming the pre-existing data graphic in the system (e.g., changing a visual attribute or visual encoding), or by importing static data graphics created in another tool and relying on the system to link the two static frames by association.

Timing: Participants considered many different options to design the timing of animations. 7 participants [50.0%] wanted to order glyphs in a list interface to specify delay based on data or visual attributes as seen in P10's sketch in Figure 4.1.3. 6 participants [42.9%] described using a formula editor interface to set duration or delay based on computed functions from data or visual attributes such as P5's drawing in Figure 4.1.2. While 6 participants [42.9%] described the need to select timing effects from a list of presets & templates such as the "domino effect" described by P4 to stagger glyphs' delay. 4 participants [28.6%] described using relational timing methods such as "animate after" or "animate with" to stage animations.

Working with data: 5 participants [35.7%] described the need to have a data table interface for inspecting data. All but one participant [92.9%] described the system concept to link glyphs by their underlying data tuples or data transforms. Participants also described the desire to have system concepts to assist with data-driven generation: 11 participants [78.6%] described auto-generation of visual encodings, 6 participants [42.9%] described the need for the system to provide highlighting or filtering options based on a selected data attribute, while 6 participants [42.9%] desired the ability to sort glyphs by data attributes, and 5 participants [35.7%] described grouping glyphs by a categorical data attribute.

4.4 System: Design Objectives and Overview

In this section, we introduce design objectives that inform the conceptual framework and user interfaces of Data Animator. These design objectives are in part based on the findings from our ideation study in Section 4.3. We then discuss what kind of knowledge an animation authoring system should assume about static visualizations and substantiate Data Illustrator as the input data graphics format. Finally, we briefly summarize the Data Animator system and introduce an example data narrative about urbanization in East Asia [154].

4.4.1 Design Objectives

The design objectives (**DO**) target an authoring system for users with a design background and minimal programming experience. We derive the following objectives from our ideation study on how designers conceptually approach authoring animated data graphics in Section 4.3.

DO1: Focus on keyframe animation, yet introduce additional authoring paradigms when advantageous. Results from the ideation study in Section 4.3 indicate that designers prefer keyframe animation because it is the paradigm they are most familiar with. The participants also noted that combining multiple paradigms to strike a balance between ease of use and control. In Data Animator, we use keyframing as the primary authoring paradigm, where animating from one data graphic to another is achieved by treating the source and destination data graphics as two keyframes and then tweening the differences between them. We also try to identify scenarios where keyframing may become tedious and introduce additional paradigms in these cases.

DO2: Augment familiar design concepts from existing animation tools. Similar to **DO1**, our objective is to leverage concepts from existing graphic, animation, and prototyping tools. This approach can improve the system's learnability by re-using concepts and user interfaces that we reasonably assume designers to be familiar with. When necessary,

we augment these concepts to support data graphics and aim to interweave them within a fluid authoring experience.

DO3: Promote automated yet flexible matching of objects in a transition. One of the challenges in animating data graphics is to specify the matching between objects' in the source data graphic and those in the destination data graphic. We aim to design an automated matching method to reduce the burden on designers. Since a fully automatic approach may not be perfect, it is also important to clearly present the matching results to the designers and provide them the flexibility to fix errors and make adjustments.

DO4: Compose relative timing components to author expressive pacing techniques. Authoring meaningful animations in data graphics requires support for pacing techniques such as staging, staggering, or speed variation. These techniques prescribe that the triggering and duration of an object's animation depends on the animating behavior of another object. For example, staging breaks down a transition into sub-transitions that start animating after the previous animation stage ends. These relative timing components are often determined by the data attributes bound to the objects, or the hierarchical relationships between the objects in the visualization. It is our goal to bridge the gulfs of execution and evaluation [155, 156] in authoring these timing components.

4.4.2 Assumptions about Static Visualizations

We consider the authoring of *static* visualizations out of the scope of this work, because it is a well-studied area with many tools available to use [22, 27, 23, 35]. A prerequisite for using Data Animator is thus to prepare static visualizations that can be used as keyframes. A variety of formats for static visualizations are available, such as raster images, scalable vector graphics (SVG), and other proprietary formats used by different authoring tools. The format of a static visualization has significant implications on how it needs to be imported and processed. For example, a raster image contains no information about the marks, their properties and data bindings, and would require sophisticated computer vision techniques

to extract such information.

Since our focus is on authoring animated transitions, we need a format the describes the following pieces of information about a static visualization:

- **Marks**: properties of all the graphical objects in a visualization including shape, geometry, and visual styles such as fill color, stroke width, stroke color, and opacity.
- **Hierarchical Organization**: in a visualization, marks are often grouped to form higher-level constructs (e.g. a glyph may consist of multiple marks [157]; multiple marks may form a collection (e.g., stacked bar chart). The format should describe such hierarchical relationships between graphical objects clearly.
- **Peers**: given selected a mark or a glyph, we want to find its peers, which refer to all the other instances that are generated together with it. For example, in a scatter plot, the peers of any circle are the other circles representing the same type of data items. Being able to get the peers of an object is important because when a visualization consists of multi-class marks, it is easy to separate these classes.
- **Data Scopes**: for each graphical object (low-level marks or high-level constructs) that represents data, the format should describe what data rows are bound to the object as its data scope [35, 157].
- **Visual Encodings**: for each visual property that encodes data, the format should specify which data attribute is encoded.
- **Scales and Axes**: for each visual encoding, the format should record the associated scale information, including scale type (e.g., linear, logarithmic), the domain and the range. Axes or legends are graphical manifestations of scales. It is desirable to have their information such as positions on screen, because we also want to support animated transitions of axes and legends.

Based on these requirements, we examined a few visualization formats. The SVG format records information about marks and sometimes hierarchical organization, but extra efforts are needed to infer data scopes and visual encodings [158]. The dSVG approach used in Canis [99] requires users to manually define if two objects from different static visualization are representing the same data through the ID and datum tags, but visual encoding or scale information is not included.

We chose the Data Illustrator framework and its associated file format as our input visualization specification. Starting with a mark, Data Illustrator uses the *repeat* or *partition* operators to generate its **peer** marks, and bind data rows to each mark as its **data scope**. Repeat and partition can be concatenated multiple times to create **hierarchical structures**. Users can also bind data attributes to visual properties, and the system automatically generates **scales and axes**. Throughout the authoring process, Data Illustrator records all the required information in its static visualization output. Data Illustrator uses the JSON file format to represent all the information. The supplemental materials contain sample Data Illustrator files that can be imported as vis boards, and they are using the ".diproj" extension.

4.4.3 Overview

Data Animator allows users to create animated data graphics from sequences of visually diverse data graphics. The tool consists of three views for authoring animated transitions at different levels of granularity:

- **Storyboard View** (Figure 4.2 - top): Here users import static visualizations created in Data Illustrator as *vis boards*, which are considered as keyframes. Users can sequence the vis boards in a 2D workspace. This design is inspired by storyboarding, a technique applied in film production and UX prototyping [30, 31, 127, 20]. *Vis boards* can be arranged, duplicated, or removed from the project. To create an an-

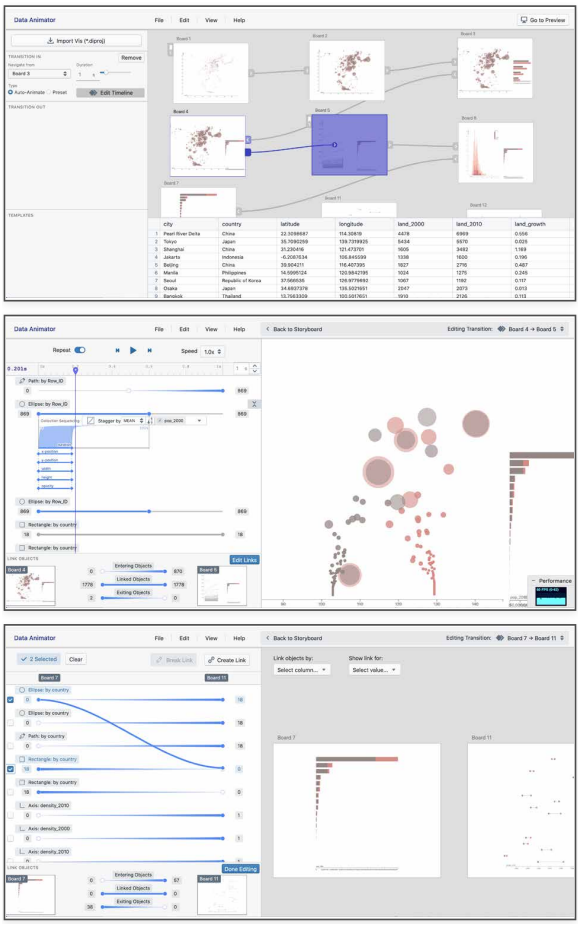


Figure 4.2: The main interfaces of Data Animator: Storyboard View (top), Timeline Editor (middle), Object Matcher (bottom).

imated transition, the user clicks and drags a board's connector to then drop on a desired *destination vis board*. Data Animator automatically generates a transition based on an object matching algorithm. All object timings are set to defaults that can be overridden in the "Timeline Editor".

- **Timeline Editor** (Figure 4.2 - middle): Clicking on any connection between two vis boards in the Storyboard View will switch to a Timeline Editor, where users can view how objects across two vis boards are matched through an automated algorithm. They can also manipulate the timelines to pace temporal rhythm of animating objects.

- **Object Matcher** (Figure 4.2 - bottom): If the matching of visual objects does not align with users' expectation, they can invoke the Object Matcher View through a

button and tweak the automated results of object matching.

In the next two sections, we explain the working mechanisms of Data Animator in detail. We will use a data narrative about the urban population growth in East Asia from 2000 to 2010 (referred to as "Urbanization") as a running example. We base this example on the award winning project by Nadieh Bremer [154]. The dataset contains 869 rows for each urban city in eastern Asia, with twelve columns including geographic details (e.g., `city`, `country`, `latitude`, `longitude`) and population growth quantities (e.g., `2000_population`, `2010_population`). The animated data graphics in this example transition through nine unique visualizations – Figure 4.3 features all eight animated transitions. The story deftly moves from maps of each urban city (Figure 4.3.a) to introduce auxiliary bar charts (Figure 4.3.b-c) to slope charts (Figure 4.3.d) to histograms (Figure 4.3.e) then zooms in to bar charts that compare growth by country (Figure 4.3.f), and finally drives home how economic growth of each country is tied to urbanization with a connected scatter-plot (Figure 4.3.g-h). The video illustrating all of these animations is at http://data-animator.com/gallery/urbanization.html.

Section 4.5 addresses the challenge of coordinating visual objects between transitioning data graphics by detailing the object matching algorithm of the framework. We then describe the Object Matcher, a novel interface for users to visualize and manually disambiguate the matching between two data graphics. Section 4.6 details how Data Animator empowers designers to pace the temporal rhythms of a transition through the Timeline Editor.

4.5 System: Coordinating Objects in Transition

To generate an animated transition from a *source vis board* to a *destination vis board*, Data Animator needs to know how the objects (e.g., marks, glyphs, collections, axes & legends) in these two vis boards should map to each other. Even though we have all the information about graphical objects and data bindings in a single static visualization, automatically

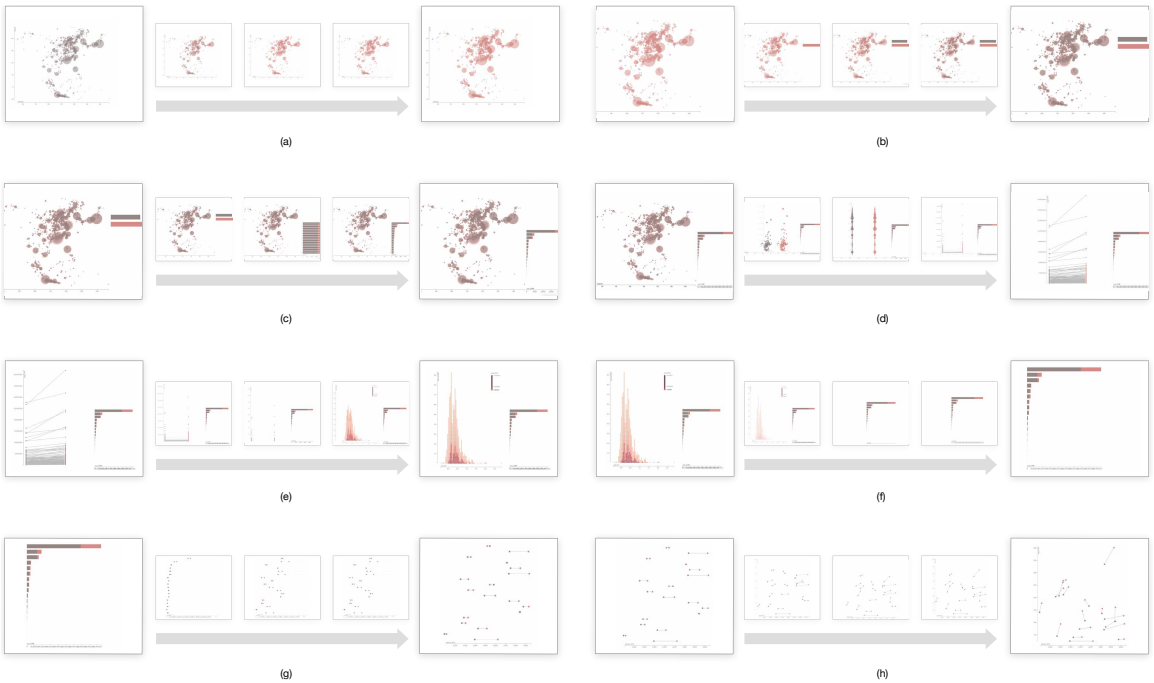


Figure 4.3: Still frames of featured animated transitions from "Urbanization" example scenario. (a) City dot map to symbol map of population for 2000 and 2010; (b) Introduce summary bar chart of all cities to right; (c) Change bar chart to compare by country; (d) Symbol map transitions to slope chart for population growth from 2000 to 2010; (e) Slope chart to histogram for % population growth; (f) Exiting histogram to scale in summary bar chart; (g) Rectangles of bar chart change to circles of dumbbell plot; (h) Introduce y-axis to transition to connected scatter-plot.

matching the objects across two vis boards is non-trivial.

In the simplest case, there is a one-to-one mapping between the objects in the two vis boards. In Figure 4.4.a, each circle in the *source* vis board (left) matches to an area-encoded circle in the *destination* (right) when they have the same data scope, that is, both are bound to the same data rows with the same city value and the same country value. During the animation, the differing visual property values of matched objects are interpolated to form a transition – for example in Figure 4.4.a the `area` of the matched circles change in the animated transition. The matching applies not only to marks but also other visual object types such as axes & legends.

There is not always a one-to-one mapping between objects in the source and destination vis boards, however. Objects can merge or split between vis boards. For example, in

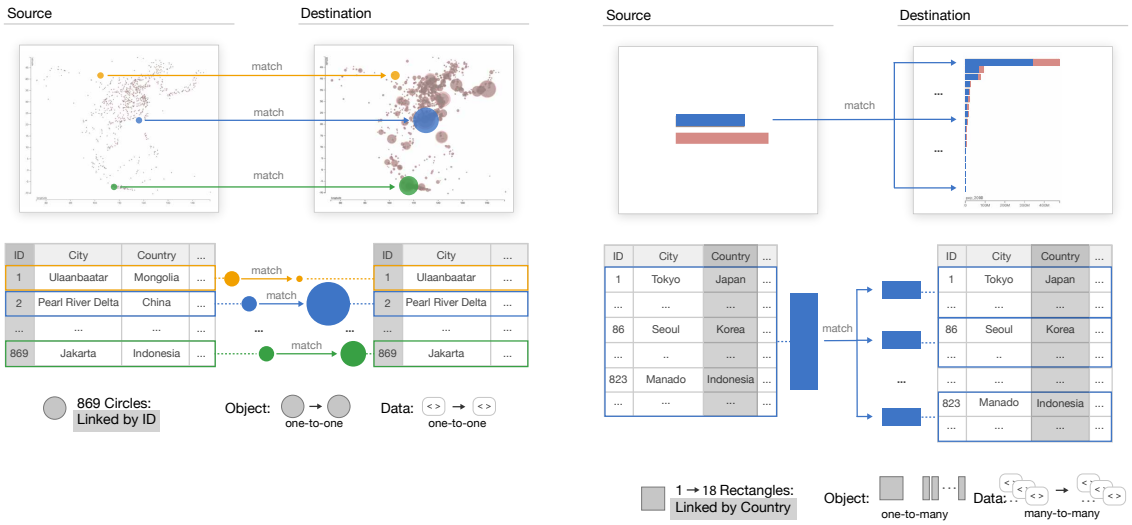


Figure 4.4: (a) Example of one-to-one matching from transition in Figure 4.3.a. Circles linked by `ID` with one-to-one object and data matches. (b) Example of one-to-many matching from transition in Figure 4.3.c. The data scope of the blue bar in the source vis board is the union of the data scopes of all the blue bars in the destination vis board.

Figure 4.4.b, there are two bars in the source vis board, representing year 2000 and 2010 respectively. In the destination vis board, we have two superimposed bar charts, where each row represents a country, and the two bars in each row represent the two years 2000 and 2010. The data scope of the blue bar in the source includes all the data rows where the year is 2000. In the destination vis board, the data scope of each blue bar is a data row where the year is 2010 for a specific country. The union of the data scopes of all the blue bars in the destination thus equals to the data scope of the blue bar in the source. In the animated transition, it thus makes sense to create a one-to-many matching, where the blue bar in the source splits into multiple blue bars in the destination. In such cases where a split or a merge is required, a preset effect is applied to specify that the aggregated shape *repeats* (seen in Figure 4.4.b) or *partitions* to make up the counterpart shapes in the other *vis board*.

In other cases, some objects in a vis board may simply not have any matching counterparts in the other vis board. These unmatched objects need to enter or exit the scene, and they can be animated based on a preset effect such as “Fade In” or “Fade Out” (the default

presets). Preset effects create a desired animation by supplanting an object's property value (e.g., `opacity=[0]` for "Fade In").

4.5.1 Automated Object Matching

To handle these different cases, our overall approach of matching objects operates at the level of *object sets* instead of individual objects, and computes a *matching score* between two object sets based on a number of predefined criteria.

Given a source vis board and a destination vis board, we first group all the objects into *object sets* for each of the boards. An object set refers to a set of peer marks, peer groups or peer collections. For example, in a scatter plot, all the marks form an object set; in a trellis view of bar charts, at the highest level, we have an object set of bar charts, within each bar chart, we have an object set of rectangles. The object sets are identified through the peer information (represented as *class ID*) and the hierarchical structures recorded in a Data Illustrator file. For axes and legends, each axis or legend is an object set.

After identifying the object sets, we first check the types of members for every pair of object sets, one from the source vis board, the other from the destination vis board. By definition, the members of an object set are peers of each other and have the same type. For marks, the types can be rectangle, ellipse, path, or text. For collections, the type is either a repeat grid (members are created using Data Illustrator's repeat operator) or a partition (members are created using Data Illustrator's partition operator). For axes, the type refers to the data attribute type (categorical, quantitative or temporal). For legends, the type refers to the visual property (e.g., size, continuous color, categorical color). If two object sets have different types of members, we determine that these two object sets do not match. Otherwise, we proceed to the next step.

For a pair of object sets sharing the same type of members, we compute a matching score between them. The matching score is the weighted sum of the following components, summarized in Table 4.2.

Table 4.2: Components of a Matching Score between two object sets S_1 and S_2

Component	Explanation	Scoring Function	Weight
cardinality	the number of members in the object set	$\frac{abs(\|S_1\| - \|S_2\|)}{max(\|S_1\|, \|S_2\|}$	3
populating field value	the categorical attribute used in repeat or partition	Let P(S) be the union of all field values in object set S $\frac{abs(\|P(S_1)\| - \|P(S_2)\|)}{min(\|P(S_1)\|, \|P(S_2)\|)}$	2
data scope	data rows bound to each member of the object set	let $D(S)$ be the data scope of object set S $\frac{abs(\|D(S_1)\| - \|D(S_2)\|)}{min(\|D(S_1)\|, \|D(S_2)\|)}$	2
shape ID	unique identifier assigned to each mark	let $U(S)$ be the union of shape IDs in object set S $\frac{abs(\|U(S_1)\| - \|U(S_2)\|)}{min(\|U(S_1)\|, \|U(S_2)\|)}$	1.5
class ID	identifier assigned to each mark to identify its peers	let $C(S)$ be the union of class IDs in object set S $\frac{abs(\|C(S_1)\| - \|C(S_2)\|)}{min(\|C(S_1)\|, \|C(S_2)\|)}$	1.5

- **cardinality**: the number of members in an object set. Two matching object sets need not have the same cardinality. For example, the destination vis board may remove some marks in an object set from the source vis board in order to achieve an animated filtering effect; or a shape may be splitting into multiple shapes. Instead of using a binary scoring function, we compute the score of cardinality as the percent difference between the cardinalities of two object sets.

- **populating field value**: when an object set is a collection (i.e. a repeat grid or a partition), Data Illustrator records which field or data attribute was used to perform the repeat or partition operation to populate its members. For example, in Figure 4.4.a, the circles in the scatter plot were created by repeating an initial circle using the ID attribute. Each of the circles created is assigned a populating field (ID) and value (an ID value). The matching score for populating field & value is computed first by doing a union on the field & values for all the members in an object set, then

calculating the percent difference between the two object sets.

- **data scope**: as mentioned in Section 4.4, data scope refers to the data rows bound to a graphical object. For example, in Figure 4.4.a, we have an object set consisting of the circles in the scatter plot. Each circle in the scatter plot in the source vis board has a data scope of exactly one row. The data scope of the object set, then, is the union of the data scopes of all the circles. The matching score for data scope is computed as the percent difference between the data scopes of two object sets.

- **shape ID**: when the vis board was created, Data Illustrator automatically assigns a unqiue ID to each shape. These IDs can accurately reflect how the same Data Illustrator file evolves over time, if the designer saves different states of the design as multiple vis boards over the course of authoring.

- **class ID**: id from Data Illustrator that is used for all peer shapes, this information helps us know if marks or collections are a part of same object set. The matching score is computed in the same way as the shape ID.

The overall matching score between two object sets is the weighted sum of all the above component scores. Table 4.2 shows the weights we assign to each component. These weights are designed based on the following considerations: cardinality, populating field value and data scope are three most indicative criteria to determine if two object sets are representing the same data. They are thus assigned the most weight. Shape ID and class ID are less reliable, for example, if designers worked on the source and destination vis boards in different sessions, the IDs may not match. They are thus assigned less weight. For a given object set in the source vis board, we pick the object set from the destination vis board with the highest matching score. If that maximum score is greater than or equal to 5 (out of 10), we designate this pair of object sets to be matching. Otherwise, there is no matching object set from the destination vis board.

We use a threshold of 5 to determine if there is a match based on a few plausible matching scenarios: whenever the cardinality and the data scope scores are both 1, or the cardinality and the populating field value scores are both 1, or the populating field value, the data scope and shape ID scores are all 1, we can say with high confidence there is a match. In all these cases, the weighted sum would be greater than or equal to 5 (out of 10).

4.5.2 Visualizing Object Matching Results

Data Animator enables designers to view the matching results in the Timeline Editor. As mentioned in Section 4.4, automated matching is performed when users connect two vis boards in the Storyboard View (Figure 4.2 - top). To view the matching results between two vis boards, users select the connection linking these boards, and click the ”Edit Timeline” button. The interface displays the Timeline Editor (Figure 4.2 - middle). The left panel (Figure 4.5) shows the results of automated object matching for the transition in Figure 4.3.a. In this transition, the source vis board shows a bubble plot of the populations of different cities in 2000, and the destination vis board shows a bubble plot of the populations of different cities in 2010. In both visualizations, the x axis is the longitude of the cities, and the y axis is the latitude of the cities. The bottom part of the panel shows a preview of these two vis boards. The main area of the panel displays multiple *layers*, one for each pair of matched object sets. For example, the ellipses in the source and destination vis boards match up perfectly: the cardinality of the ellipse object set is 869. Such a perfect matching is represented as a timeline with the same thickness at both ends. Since the visual properties (size and fill color) of the matched ellipses are different in the two vis boards, the timeline is colored blue, indicating tweening is needed. In this case where automated object matching is successful, the animated transition as a result of interpolating the ellipses works without any user intervention. The longitude and latitude axes match up perfectly too. In addition, there are no differences in their visual properties. Their timelines are thus colored gray, indicating that no tweening is needed.

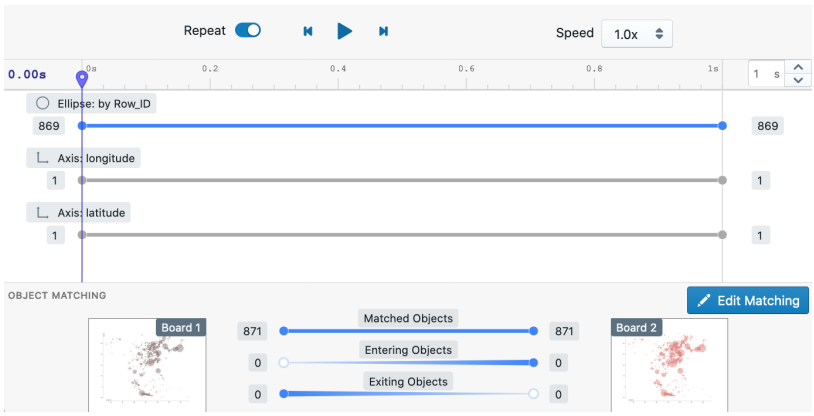
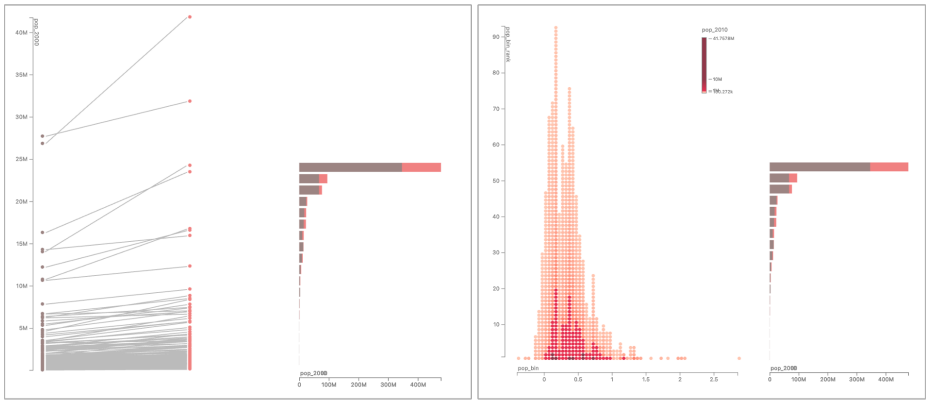


Figure 4.5: The left panel of the Timeline Editor, which shows the results from the automated object matching for the two vis boards in Figure 4.3.a. The first timeline specifies the behavior of 869 ellipses and the final two correspond to the x and y axes.



Source vis board Destination vis board

Figure 4.6: Enlarged view of the source and destination vis boards for the animation in Figure 4.3.e.

For the transition in Figure 4.6 (corresponding to Figure 4.3.e), more object sets are involved. In the source vis board, we have a slope graph showing the changes in urban population from 2000 to 2010 (each line is a city, and the left anchor points represent year 2000, the right anchor points represent year 2010), and two bar charts superimposed over one another (each bar represents a country, the brown bars represent urban populations per country in 2000, and the red bars represent urban populations per country in 2010). In the destination vis board, the bar charts remain the same, but instead of a slope graph, we have a dot plot. The horizontal axis represents binned growth factor of population, and each dot represents a city. The color of the dots represents the city's population in 2010. The

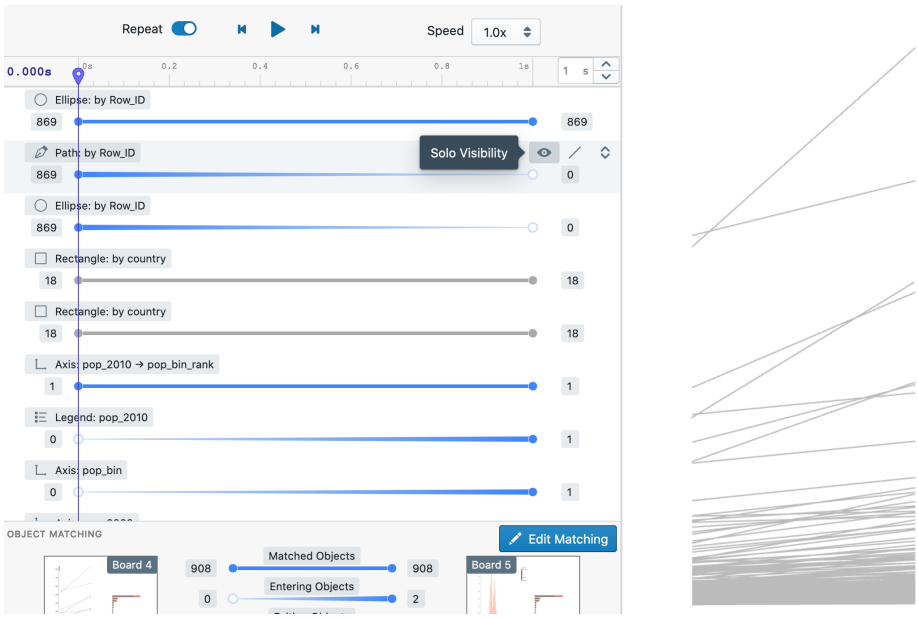


Figure 4.7: Hovering the mouse over the eye icon will show only the corresponding object set in the preview on the right.

transition thus primarily happens between the slope graph and the dot plot (Figure 4.6).

In this example, there are many layers, each representing the matching result for one pair of object sets. Since it might be difficult for users to understand which object set each layer corresponds to, we added a feature to let users filter out object sets by hovering over the eye icon at the top right corner of each layer. For example, in Figure 4.7, the preview area on the right only shows the path object set in the slope graph. This path object set does not have a matching set in the destination vis board, because the slope graph is replaced by a dot plot. The timeline for the path object set thus has 0 thickness at the right end. This tapered representation shows that the path objects will be exiting in the destination vis board. Conversely, the legend for the dot plot (Legend:pop_2010, seventh layer in Figure 4.7) is absent in the source vis board, and it will be entering into the destination vis board. The timeline representation thus is tapered where the left end has no thickness.

4.5.3 Adjusting Matching Results through the Object Matcher

Since automated matching may fail in certain cases, or produce confusing transitions, Data Animator provides the Object Matcher view, a novel user interface for designers to interpret and adjust the matching results. Wrongly matched objects can be unmatched, and users can also manually create a match between two or more sets of objects.

Using the example in Figure 4.6 again, there are two sets of ellipses in the slope graph in the source vis board: the set on the left colored in gray represents cities in 2000, and the set on the right colored in red represents cities in 2010. In the destination vis board, there is only one set of ellipses in the dot plot. All these three sets have the same cardinality (869), and it would be a perfect match to map any of the two sets in the slope graph to the set in the dot plot. In this case, Data Animator creates a matching between the red ellipses in the slope graph and the ellipses in the dot plot (first timeline in Figure 4.7). This also leaves the gray anchor points without any match (the third layer in Figure 4.7). Logically this matching is correct, but the resulting animated transition looks odd as the red ellipses move through the center of the chart and cross over each other to get to their position in the dot plot. Designers may prefer instead having the slope graph fade out and the dot plot fade in, to reduce the number of flying objects on screen. Thus, the existing match needs to be removed.

To remove a match, users click the “Edit Matching” button in the lower part of the panel (Figure 4.7) to switch to the Object Matcher View (Figure 4.8), which is a modified version of the Timeline View. Users can select the matched object sets (in this case the first layer), and the preview area on the right shows how individual marks match across the source and destination vis boards using a blue link (Figure 4.8). After selecting the matched object sets, users click the “Unmatch” button at the top to break the matching. With the matching removed, users can go back to the Timeline View by clicking “Done Matching”. They can control how the ellipses in the slope graph animate out (have the slope graph ellipses fall down with the ”Move Out - to Bottom” preset) and then have the dot plot ellipses rise up

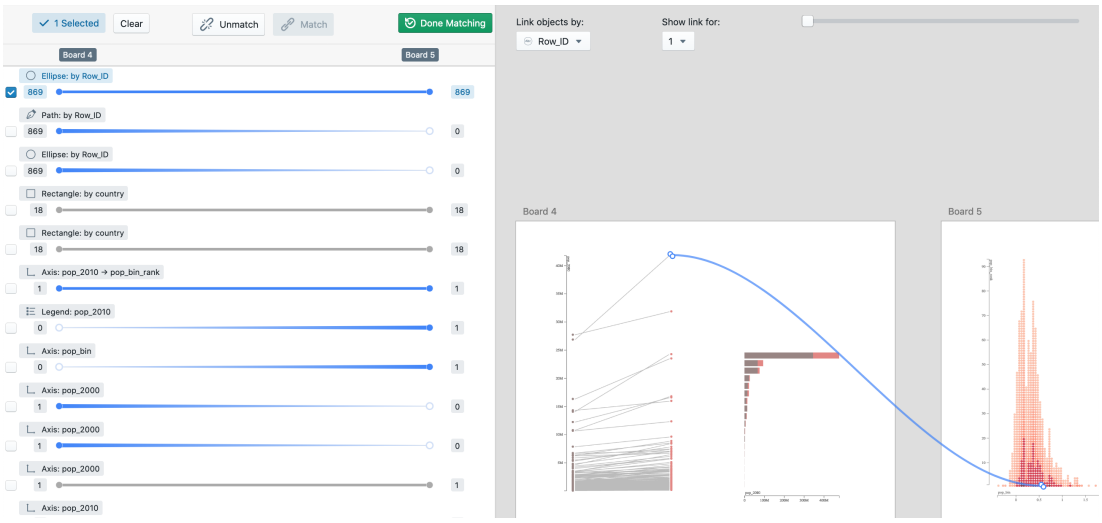


Figure 4.8: Selecting a matching result previews the links between objects.

with the ”Move In - from Bottom” preset (Figure 4.9).

In another example where we need to transition from a superimposed bar chart to a dumbbell chart (Figure 4.10), the bar chart in the source vis board superimposes two bar charts, representing the population for each country in 2000 and 2010 respectively; the dumbbell chart shows the same information with different encodings: each country is a path, and the x position of the ellipses represents the population in 2000 and 2010 respectively. Data Animator does not make a match between any of the marks, because their types do not match at all. However, a match will be beneficial, as the animation maintains congruency so that the two sets of bars transition into the two sets of ellipses.

To manually create the matching, users click the “Edit Matching” button to switch to the Object Matcher View (Figure 4.11). Users select the two object sets they want to link, and then click the “Match” button. Data Animator will create a matching between the two sets and generate intermediate frames by tweening.

We refer the reader to the accompanying video that more fully illustrates the dynamics of these specifications and flow through the user interface.

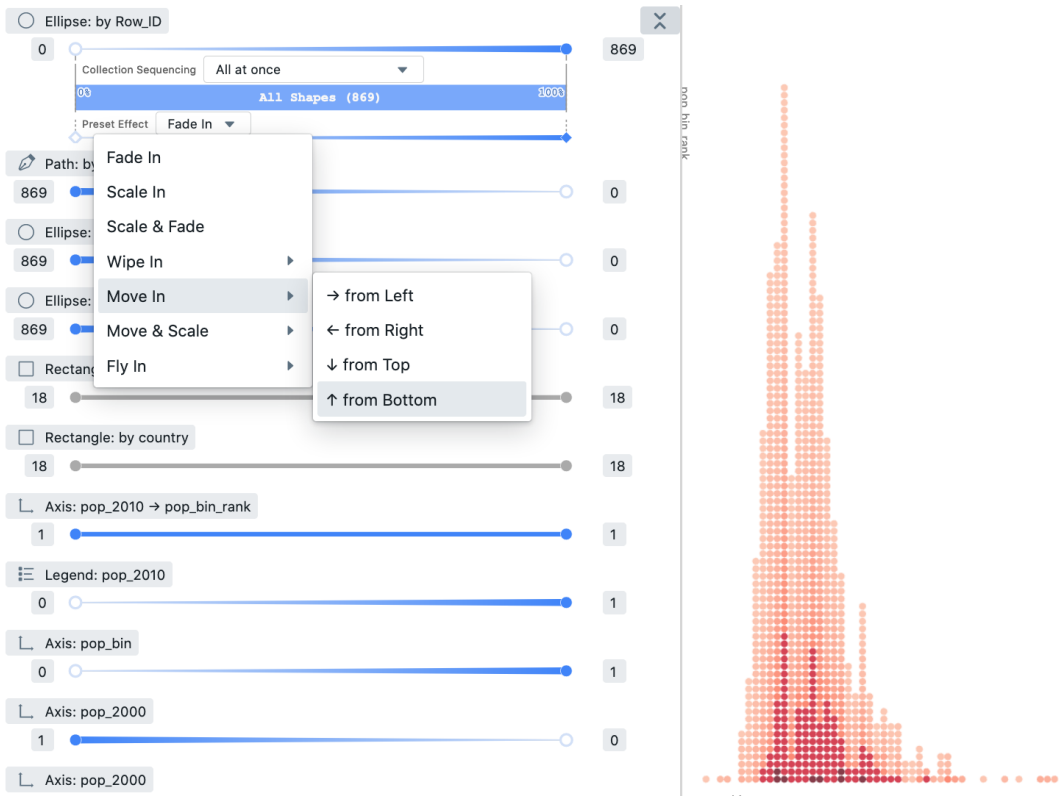
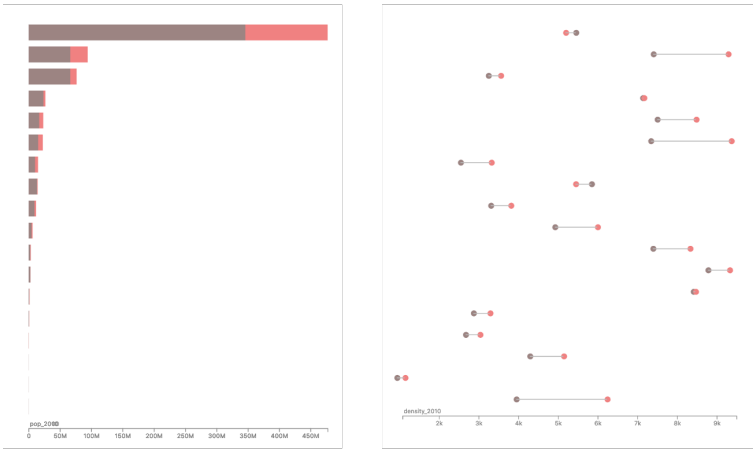


Figure 4.9: Interface controls for choosing an entering or exiting preset effect for an unmatched object set.

4.6 System: Specifying Temporal Pacing of Animations

After matching the objects in the source vis board and the destination vis board, automatic animated transition through tweening does not always achieve desired results. Animated data graphics often contain hundreds or more graphical objects, and it can be overwhelming for all these objects to start animating at once.

Data Animator supports the division of a complex animation into stages and uses data attributes to stagger the start time and vary the speed within sets of animating objects. The "Timeline Editor" in Data Animator differs from other timeline interfaces in design tools [31, 29] by allowing users to edit temporal properties of object sets, rather than tediously creating and adjusting each keyframe for tens or hundreds of objects. To get rapid feedback on the current design, users can pause and play the animation preview, change the



Source vis board Destination vis board

Figure 4.10: Enlarged view of the source and destination vis boards for the animation in Figure 4.3.g.

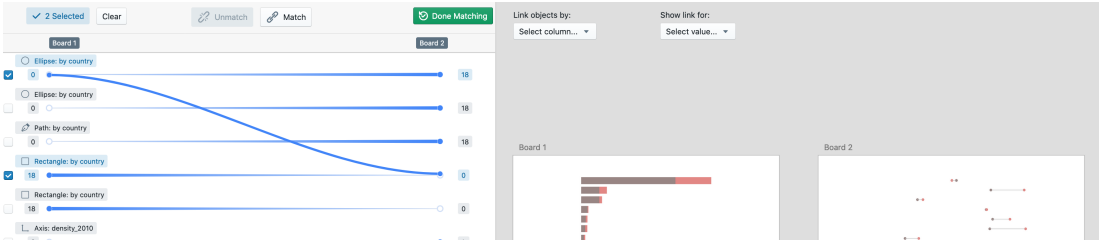


Figure 4.11: Selecting two unmatched object sets to create a matching.

rendering speed (e.g., "0.5x" for half speed), and manually scrub the playback by clicking and dragging the playhead (Figure 4.12).

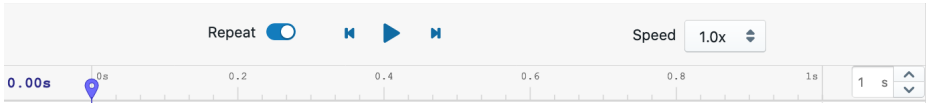


Figure 4.12: At the top of the left panel in the Timeline Editor, users specify the duration of animated transition between two vis boards

In this section we describe the temporal concepts of the system framework and introduce a novel timeline interface for designers to pace the temporal rhythm of data-driven objects.

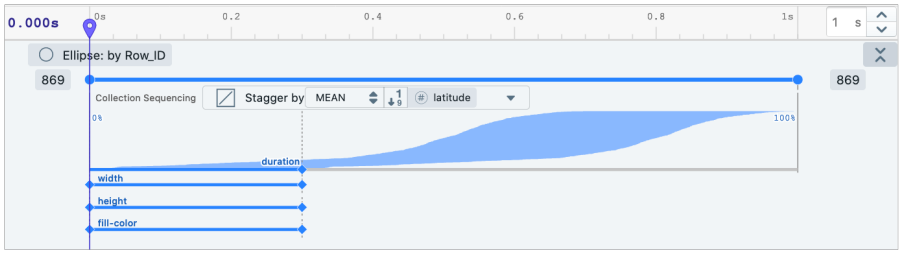
4.6.1 Staggering and Speeding by Data Attribute

In a simple example like Figure 4.5, where the size and color of the ellipses in a scatter plot animate between two vis boards, designers may want to stagger the animation by introducing an incremental delay to the starting time of each ellipse. Instead of having to do so manually for all the 869 ellipses, they can use the ”Expand” button in each timeline to show the user interface controls related to temporal pacing. By default, all the ellipses start at the same time with the same duration (Figure 4.13 (a)). Users can choose from the drop-down menu to either stagger the starting time or control the speed by an ordinal or quantitative data attribute. For example, users may want the cities at lower latitudes to animate first, so they can choose staggering by the attribute “latitude”. The timeline visualizes how staggering will work by plotting a horizontal line for each ellipse, where the left and right ends of the line represents the starting and ending time of the animation. These lines form an overall shape depicting the distribution of starting and ending times of the ellipse set (Figure 4.13(b)). Alternatively, users may want to control the speed of animation by data, for example, by letting cities with greater population growth to animate more slowly. They can select “speed by pop_growth” from the drop down menu, and the timeline shows a visual summary of the effect of this action (Figure 4.13.c). When users stagger or speed by a quantitative attribute, they can also choose the aggregate (e.g., mean, max, min) if the object’s data scope consists of more than one data row and has more than one value for the chosen attribute.

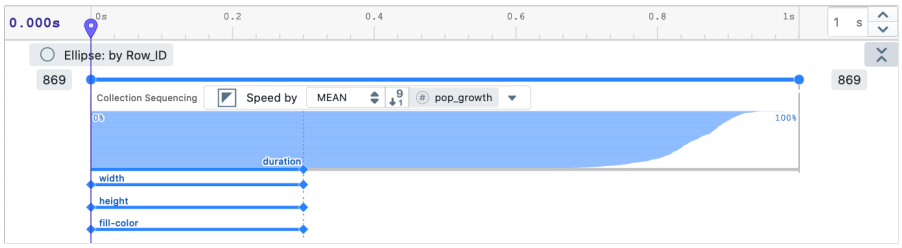
4.6.2 Staging

Staging is another commonly used technique to break down a complex animation into sub temporal groups. In the example in Figure 4.5, multiple visual properties are animating: size (width and height) and fill color. Users may want to animate the size properties first before animating the fill color. In Data Animator, each animating property has its own layer (Figure 4.13.a). To create staging, users simply drag the end points of a timeline in a layer

(a) All objects start animating at the same time with the same duration by default



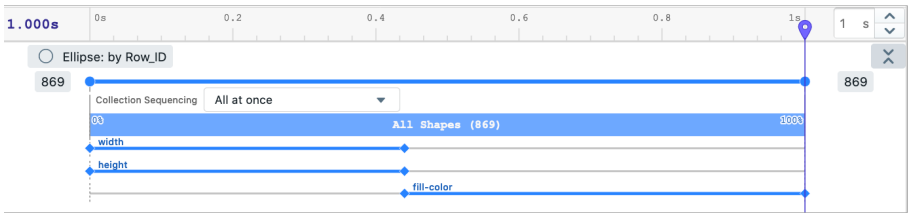
(b) Staggering objects by the data attribute “latitude”



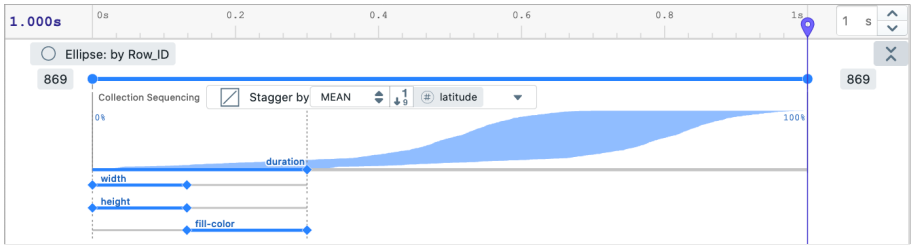
(c) Set the speed of animation objects by the data attribute “pop_growth”

Figure 4.13: Interface controls for using data attribute values to modify the speed and staggering of animation effects. The middle modification makes the animation start times delay (stagger) as a function of the latitude data attribute. The bottom modification changes the speed to be a function of population growth.

to change the starting and ending time of the animation for a property. In Figure 4.14 (a), two stages are created. In the first stage, the width and the height properties animate; in the second stage, the fill-color property animates. Data Animator also supports applying staggering after the stages are set up. In Figure 4.14 (b), we use the “latitude” data attribute to stagger the ellipses after the stages have been created in Figure 4.14.a. Due to staggering, each ellipse will have less duration to animate, and the resulting durations for the two stages are allocated proportionately according to the original staging design. Figure 4.14.b shows how this information is conveyed in the timeline view.



(a) Two stages of animated transition: width and height first, fill-color next



(b) Staggered objects will keep the stages for the visual properties proportionately

Figure 4.14: Making modifications to create animation stages. In the top, the user creates a first stage of width and height change, followed by the fill-color changing. In the bottom view, when each of animations occurs (staggered start) can be set as a function of data.

4.6.3 Hierarchical Keyframes

Figure 4.14.b is an example of a novel concept we introduced in Data Animator: *hierarchical keyframes*, where the keyframes in a transition are constrained to the duration of its parent. The allotted time for each successive child cascades as a linear function of the parent's duration (a percentage value). There are three types of parent-child relationships in hierarchical keyframes: Object-Property, Transition-Object, and Object-Object.

- **Object-Property** - Objects with transitioning properties act as the parent timing for those differing property values. With percentage-based keyframes, the animation of properties can be staged one after another. Figure 4.14 (b) is an example of the Object-Property hierarchical keyframes, where the properties (as the children) inherit the allotted durations from the object (as the parent).

- **Transition-Object** - For a transition between two vis boards, multiple object sets may be involved. A global duration is specified as the root of the timing hierarchy. As shown in Figure 4.12, the default global duration is 1 second. When users change

the duration, the axis updates accordingly. This global duration serves as the parent timing for all of the objects' transitions between the two vis boards, and all child objects' duration is expressed as a percentage of this global duration.

- **Object-Object** - Groups or collections from Data Illustrator are higher-level constructs that visually group and layout objects. They also serve as the parents of their members in hierarchical keyframes. For example in Figure 4.15, the visualization has a group of two rectangles. The parent group object has a duration of 80% of the global duration. Each rectangle's duration accounts for 50% of their parent's duration, and each rectangle is staged one after another in the group. Changes to either keyframes of the parent group will constrain the allotted time for both child rectangles while still preserving their staging. For an example of how the Object-Object relationship is used in an actual animated data graphics, please refer to the "Stephen Few's Box Plot" video in our gallery (http://data-animator.com/gallery/few_box_plot.html).

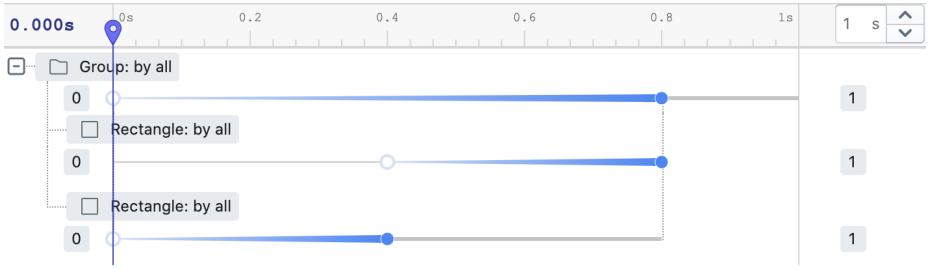


Figure 4.15: Parent objects such as groups constrain the allotted duration to child objects such as two rectangles.

4.6.4 Edit Temporal Pacing in the Timeline View: Scenarios

In this section, we present three authoring scenarios to illustrate how we can use hierarchical frames to create various staging and speeding effects.

Scenario 1: This first example is based on Figure 4.5, where the size of the ellipses animates to show data from `population_2000` to `population_2010`. The timeline

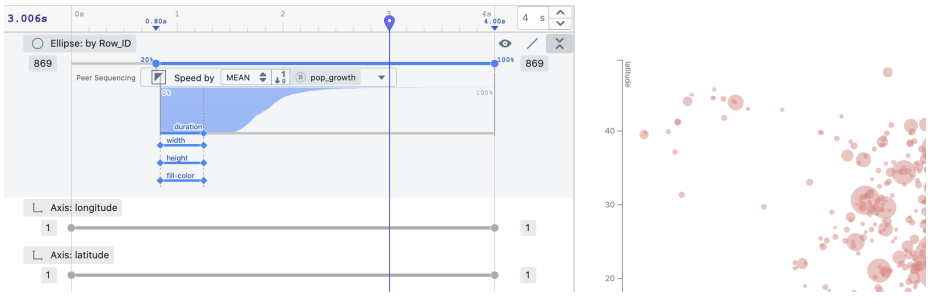


Figure 4.16: Pacing the animation for ellipses growing on a map. The user has delayed the start of the ellipses' movement and sets the speed to be a function of population growth.

consists of three layers: the matched ellipse sets, the matched x-axis for *longitude*, and the matched y-axis for *latitude*. Each layer consists of a source keyframe, a destination keyframe, and a path duration. For each layer, the object count is shown next to the source and destination keyframe (e.g., there are 869 *peer* ellipses matched from source to destination). Keyframes control the start and end times for all objects in the set to animate. In Figure 4.16, we click and drag the start keyframe to delay the set of ellipses from the start of the transition by 20%. The ellipses are now all delayed by 20%, simply dragging one keyframe sets the keyframes for all 869 ellipses in the set. In Data Animator, keyframes are assigned a percentage value rather in seconds. As mentioned in Section 4.6.3 this allows for relational timing to be preserved. When the user drags a keyframe the computed time in seconds appears on the timeline above. This feature is particularly useful for interpreting timing within hierarchies.

In this animation, by default, the ellipses animate uniformly within the set. To focus the viewer's attention on the ellipses that dramatically grow in size, we vary the speed of each animating ellipse based on the `population_growth` data attribute. Varying the speed creates an effect where the duration of each animating ellipse in the set is equivalent to how much the ellipse changes in size. To achieve this in the interface, we change the "Sequencing" from "All at once" to "Speed" by population growth rate. The sequencing updates to show the groupings created for population growth rate, and how they vary the speed (or duration). Playing the animation back, we notice it is too quick to perceive the

changes in ellipse size. So, we modify the duration of the entire transition to 4 seconds. All of the relative timings such as the delay and speed sequencing are preserved due to the hierarchical keyframe approach of Data Animator.

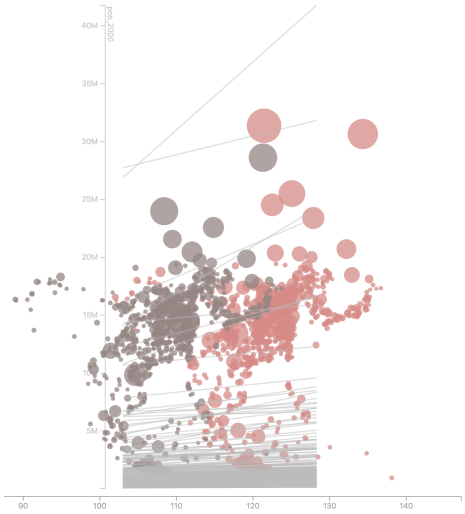


Figure 4.17: Transitioning from a symbol map to slope chart without staging.

Scenario 2: Animating numerous sets of objects at the same time can result in a transition that is visually jarring and distracting to the viewer. For example when transitioning from the symbol map to a slope chart, the default timings results in Figure 4.17. The lines of the slope chart intersect with the ellipses of the symbol map that match to the endpoints of the slope chart. In the timeline we can coordinate the pacing of this animation to instead make small changes incrementally rather than one big transition. We start out by adding a delay to when the set of paths animate in to form the slope chart, so we drag the start keyframe of this layer (third from the top in Figure 4.18) to 90%. Now that the paths our visually out of the way, we stage the matched ellipses before them. However, this staging does not solve our problem as the ellipses are animating too many visual properties at once. To stage these visual properties, we click to expand the properties of both sets of ellipses (top two in Figure 4.18). Expansion reveals the visual properties that differ between source and destination boards. Each animating property has its own pair of keyframes that we will use to stage this animation. We start by dragging the end keyframe for the x position

to 30% in order to have it happen first. Next, we continue to stage width and height after x position and then create a final stage for the y-position. We redo the same staging process for the other set of ellipses in the timeline. And now this staging creates the effect of the symbol map clearing away from the middle of the canvas and stacking on top of each other as seen in Figure 4.18 (right). This staged transition is much more clear than that in Figure 4.17. The timeline interface allows us to change the ordering of animating visual properties within sets of objects as another means to stage.

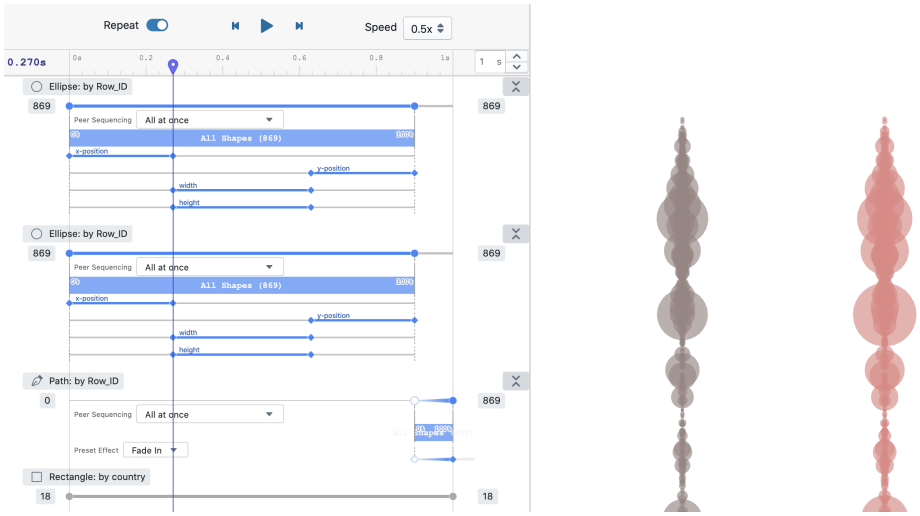


Figure 4.18: Coordinating multiple sets of unmatched objects with staging and staggering.

Scenario 3: The timeline editor also allows users to craft the pacing of unmatched objects entering and exiting from view. In this scenario, we use the example in Figure 4.6: two sets of ellipses and one set of paths that makeup a slope chart are unmatched and exiting the scene. Additionally, one new set of ellipses enters into the scene to compose the dot plot. In Figure 4.19 we create staging between the two sets of exiting ellipses by adjusting both start keyframes to 45% and the start keyframe for the set of ellipses entering to be at 55%. Now the ellipses are staged before each other with a momentary pause of 10%. Instead of fading, we want the ellipses from the slope chart to move down and out of the chart, and for the dot plot ellipses to rise up from the bottom – so we set both *preset effects* to be a move effect. Preset effects provide animation for unmatched layers that enter

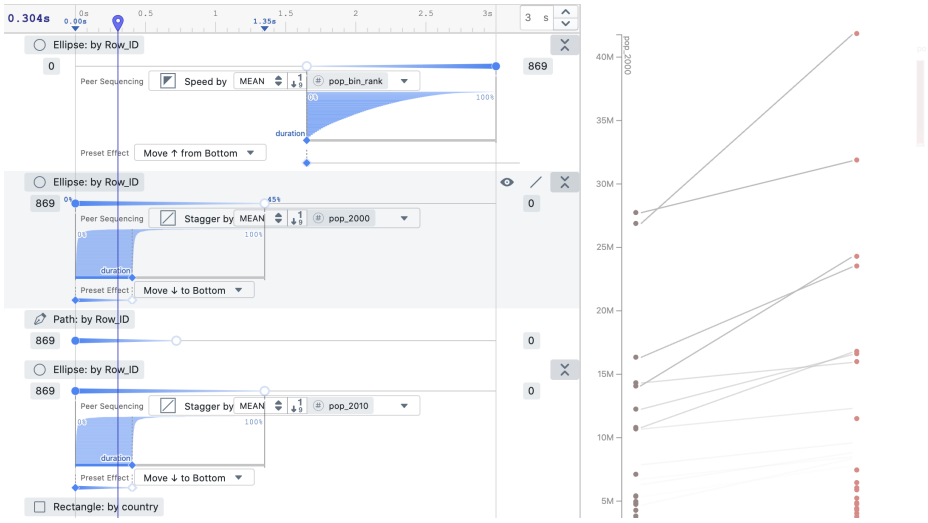


Figure 4.19: Coordinating multiple sets of unmatched objects with staging and staggering. The existing ellipses and paths of the slope chart (lower timelines) animate first and are staggered by population. The new ellipses show up after that.

or exit. Data Animator supports fade, scale, scale & fade, move, move & scale, wipe, and fly preset animations. As is, the animation has better visual ordering, but all sets of ellipses occlude each other as they move. To combat this issue, we employ staggering for both sets of exiting ellipses based on population data attributes represented in the y axis. The ellipses from the dot plot have the same occlusion problem, but instead of using staggering we opt to sequence the speed of the dot plot ellipses related to their y position. With this sequencing the ellipses now animate at equivalent speed to the distances they have to travel. This gives the visual effect of the ellipses rising together – dropping off ellipses in their respective y positions as they move upward. Finally, we need to appropriately animate the set of lines that connects the slope chart. Currently, the lines overlap with the entering dot plot, cluttering up the animation. We change the end keyframe of this set of lines to be 10% to create the illusion that the lines hold the slope chart together, and removing the lines releases the end point ellipses allowing them to fall. To time the lines with the falling end points we also sequence the lines based on the population in 2000. In this example we have gone from an undesirable transition created automatically to an animation that viewers can more easily follow with staging and sequencing of numerous sets of peer shapes.

4.7 Evaluation

4.7.1 Gallery

To demonstrate the expressivity of our approach, we created a set of animated data graphics using the Data Animator system. The collection is available at: https://data-animator.com/gallery/index.html. Each example includes a completed project file, a video demonstrating the authoring process, and the final interactive version of the animated data graphic. The gallery consists of a diverse set of re-creations and variations from projects that sample the design space surveyed in Section 4.2.

4.7.2 Usability Study

To evaluate the usability of Data Animator we conducted a re-creation study similar to the protocol used in related visualization authoring tools [35, 27]. In practice, designing animated data graphics involves gathering and cleaning a dataset, conducting data analysis to find insights, brainstorming the key visualizations and animations to convey those insights, and finally executing those ideas as an animated data graphic. A re-creation study focuses on that final execution step in the design process. For the purposes of evaluating a tool, re-creation tasks allow participants to experience authoring the same example animation and requires participants to test a larger range of the tool's features than they might otherwise. In the case of our study, re-creation tasks separate the animation authoring task from designing static visualizations in Data Illustrator. Participants are provided with completed static visualizations to import into Data Animator for each task.

Participants

For the study, we recruited 8 participants with experience in graphic, animation, and visualization design. The participants (7 male, 1 female) reside in different geographic areas in the United States and their job titles range from UX designer, graphics designer, data jour-

nalist, grad student in HCI, and professor in data governance. The participants had varying years of experience in the related design disciplines:

- Graphic Design: none (0); less than 1 year (2); 1 to 2 years (1); 2 to 5 years (3); 5 to 8 years (1); more than 8 years (1)
- Animation Design: none (1); less than 1 year (2); 1 to 2 years (2); 2 to 5 years (3); more than 5 years (0)
- Visualization Design: none (0); less than 1 year (0); 1 to 2 years (3); 2 to 5 years (3); 5 to 8 years (1); more than 8 years (1)

Participants described using animation to create interactive visualizations with animation (5) and data stories or videos (3). To create animations, the participants most frequently use programming toolkits (4), Adobe After Effects [29] (3), and presentation tools such as Microsoft PowerPoint or Keynote (4). When creating visualizations, the participants have employed a wide variety of tools such as vector editors (7); shelf builder interfaces like Tableau (6); spreadsheets such as Excel (6) or programming toolkits (5). When evaluating authoring tools with generative data support, it is important to have a mix of participants with programming and non-programming experience to see if programming knowledge affects participants' ability to understand concepts in Data Animator. Also, it is important to understand if prior experience with Data Illustrator would correlate to each participant's experience using Data Animator. The participants had varying degrees of experience with Data Illustrator: none (4); browsed the tutorials and tried once (1); used multiple times (3).

Procedure and Tasks

Each study lasted about 1.25 hours and was conducted remotely over video conferencing due to COVID-19. As a remote study, participants used their own computers with varying performance metrics, screen sizes, and operating systems. Despite the differences in computing setups there were almost no usability issues due to any of these variables (one

participant could not import through drag and drop). The only constant was that all participants used the Chrome web browser. During the session, participants shared their screens, and all audio and screen sharing was recorded. We conducted two pilot studies to test our study protocol and all the study sessions were conducted by the first author.

Participants followed a 20-25 minute tutorial led by the moderator. The tutorial walked through a running example of four animations that transition through a series of bar charts, a bubble chart, and a stacked bar chart showing medal counts from the 2012 Olympic Games in London (http://data-animator.com/gallery/olympic_medals.html).

We then asked participants to re-create six non-trivial animations from two example data stories. Participants were provided videos of each animation task and the example visualization files to import into Data Animator. The first three tasks are based on Russell Goldenberg's "Twenty Years of the NBA Redrafted" data story [122] (study version: http://data-animator.com/gallery/nba_redraft.html). Tasks 1-3 require the participant to create three animations that transition through a series of four visualizations showing how the NBA draft from 1989 to 2008 could be redrafted based on past performance to reveal which players lived up to their draft hype or exceeded expectations. The four visualizations include two scatter-plots and two bubble charts. The second example takes inspiration from Tony Chu and Stephanie Yee's "A Visual Introduction to Machine Learning - Part 1" [151] (study version: http://data-animator.com/gallery/intro_to_ml.html). Tasks 4-6 ask participants to create three animations that reveal the beginnings of a decision-tree model that searches for the best data attribute to classify housing data. The visualizations include a scatter-plot, grouped uni-variate charts, a histogram, and a bar chart. Each task was presented to the participant as a video, animation plays back and includes a description on the side. Since we were not testing the participants' skills to deconstruct an animated visualization, we explained the dataset, data encodings, and animation pacing within each example. During pilot studies we found that a video of each animation without descriptions was insufficient for participants. Participants commented that it was difficult to decipher

the pacing mechanisms of the animation. Therefore, we included text descriptions of the animation design within each video. At the end of the session, each participant completed a questionnaire and answered questions in a semi-structured interview.

The 6 re-creation tasks are meant to increase in difficulty. All participants completed each task with almost no help (except that P1 required assistance on Task 4). On the whole, participants completed the tasks quickly (μ=2 minutes 42 seconds, σ=1 minute 29 seconds). The participants completed each task in a time relative to the difficulty of the steps required (μ=average, σ=standard deviation; both in minutes:seconds):

- Task 1 - stagger ellipses based on numerical data; (μ=00:54, σ=00:28)
- Task 2 - stagger ellipses based on numerical data, stage properties within same set of ellipses; (μ=02:17, σ=00:53)
- Task 3 - stage properties for set of merging ellipses, stage entering and exiting y and x axes; (μ=02:53, σ=00:57)
- Task 4 - create a match from the set of exiting ellipses to the set of entering rectangles, stagger matched ellipses/rectangles based on numerical data, stage animating properties for matched ellipses/rectangles, stage the entering and exiting x axes; (μ=03:27, σ=01:35)
- Task 5 - stage set of rectangles before set of collections, stagger rectangles by numerical data, stagger entering collections by numerical data, stagger entering child rectangles of collection by categorical data, change preset effect for child entering rectangles, stage entering and exiting x axes; (μ=03:44, σ=01:40)
- Task 6 - stagger merging rectangles by categorical data, stage properties within merging rectangles, stage entering and exiting x and y axes, stage entering and exiting color legends; (μ=02:55, σ=01:18)

Results

Participants rated their authoring experience with Data Animator on a 7-point Likert scale. Overall participants favored the usability of Data Animator: on authoring animations, μ=2.25, σ=1.16 (1-very easy, 7-very difficult); on overall experience, μ=1.63, σ=0.74 (1-very enjoyable, 7-very frustrating); on learning, μ=3, σ=1.69 (1-very easy, 7-very difficult). As expected, the participants rated the re-creation tasks to be complex, μ=6.63, σ=0.52 (1-very simple, 7-very complex).

All participants found Data Animator to be useful for creating animated data graphics. The participants praised how Data Animator "*treats data as a first class citizen*" (P7) as it leverages the underlying data to automatically match objects between visualizations and design pacing effects based on data attributes. In particular, participants noted how Data Animator fills the gap between programming an animated data graphic and using animation design tools: "*The speed and flexibility at which you can [create animations] with Data Animator compared to writing custom code or having to wrestle with a design tool that is not meant for working with data is amazing*" (P5). The participants with programming knowledge commented how Data Animator provides comparable results to coding in a more streamlined approach "*If I wanted to create something that looked like this [with code], it would be quite a headache. Being able to change what you see and play with the animations is pretty great*" (P6). They also compared the expressiveness of Data Animator to related visualization specific tools: "*Tableau's animations are cool and magical when they get it right, but you have very little avenues for when they get the animation wrong. So I appreciate that Data Animator takes that into account*" (P7). Participants felt empowered to be able to create animations without having to program: "*As someone who doesn't code, I have to dream about creating [static] charts because there are so few tools that I can use. And then animating charts is like a dream within that dream! [Animating] is something that is completely outside of what I can expect, unless I'm willing to animate by hand in After Effects. So yes, [Data Animator] is tremendously useful.*"

The timeline interface felt natural and familiar to participants with experience using similar keyframing interfaces. The participants appreciated that Data Animator includes all of the "entry level stuff" for creating keyframe animations such as easing functions and adjusting keyframes - but also provides novel user interfaces to create staggering and speeding by data that they "*have not seen before*" (P2). All participants commented on the ease at which they could coordinate the timing of sets of objects all by changing one keyframing. Many of them described how the same process would be "time-consuming", "brutal" and "painful" to do by hand in design tools such as Adobe After Effects.

The participants also suggested additional features to perfect the timeline editing experience. Many asked for the ability to snap keyframes to nearby keyframes or adjust multiple keyframes together with selection. P5 and P7 also proposed the idea of a stage component that could divide the overall transition into sub-transitions which could alleviate the need to align keyframes when staging.

All participants were able to comprehend the results of the automatic matching algorithm. Many participants felt that this matching feature was similar to other "Auto-Animate" or "Magic Move" features from related tools [30, 113] For example P2 commented on how morphing the ellipses to rectangles in Task 4 would be "*super hard and there is no fix [in other tools] - you have to do it frame by frame in After Effects*", and they appreciated the idea of Data Animator to use data to make that matching for you. Despite understanding and appreciating the results of the automatic matching algorithm, 5 participants found the interface to manually change matches to be confusing. P7 commented that "*all the concepts are there, but it lacks the affordances in the user interface to understand what you should do*". Also, P2 commented that "*it would be nice to see what happens after you make a match – I'm guessing that was the right thing to do, but the interface doesn't immediately clarify that for me*". Future improvements to Data Animator should consider improving the "Object Matcher" interface by positioning interface elements as to guide the user in the expected order of operations, create affordances surrounding the layers to select,

clarify which action button to click next, and improve feedback when matches are created or removed.

When asked about how they would use Data Animator, the majority of participants responded that they would use it for presentation of data. P7 explains: "*my favorite visualizations takes something very complex and breaks it down piece-by-piece. So something like VORP from the NBA example, nobody gets what that is but if you take some very well-known metrics and walk them step by step to how they relate... they will at least be closer to understanding that complex concept... [I create] Keynote decks where a concept builds on top of another concept [and so on]. I usually don't use real data for [those animations], but would I ever if I could!*" Participants commented on additional uses outside of presenting data. P6 felt that it could be helpful for teaching data analysis – allowing students to create more expressive representations of their data. They also explained how a tool like Data Animator could help improve data literacy – as a broader user group (designers) would be able to create new types of visualizations. Even participants with programming knowledge felt that Data Animator would save "a lot of time" to prototype animations that they would eventually program in a fully interactive web page (P5).

4.8 Discussion

A major prerequisite of using Data Animator is that the static visualizations need to be in the Data Illustrator format. This assumption can be limiting, as Data Illustrator is not necessarily integrated in many designers' workflows. The dependency on the Data Illustrator file format does not mean that users of Data Animator need to know how to use Data Illustrator. 5 of the 8 participants in our user study have minimal experience in using Data Illustrator, but all of them could complete the tasks in a few minutes without help. We have thus found no qualitative evidence that prior experience with Data Illustrator affects the use of Data Animator. This finding suggests that Data Animator has the potential to reach a broad set of users who are not necessarily familiar with Data Illustrator.

Our primary focus is on authoring transitions between two static graphics under the keyframing paradigm. More work is needed to further enhance the expressivity of Data Animator. For example, keyframes in animations like bar chart race are generated by updating a bar chart template with a temporal attribute. Using Data Animator, creating a bar chart race would be very tedious, because designers need to prepare multiple vis boards, each corresponding to a different visualization state. Designers need to be able to import a visualization template and generate multiple vis boards automatically by a data attribute. A procedural paradigm would be appropriate for such functionality, as in the case of *transition_time* and *transition_states* in gganimate [98].

In addition, animated data graphics can benefit from techniques such as adding highlights and annotations, dynamically filtering objects by data, and syncing animated transitions with text explanations and voice narration (discussed further in Section 5.5). We plan to add these features to Data Animator in the future.

CHAPTER 5
DISCUSSION AND FUTURE WORKS

5.1 Reflecting on our Assumptions

At the outset of this research endeavor, I defined three research questions (`RQ1`, `RQ2`, `RQ3`) to guide the design, implementation, and evaluation of these visualization authoring tools. Starting with those research questions, and throughout this process, our team has scoped this problem into a manageable form. By scoping the problem, we made assumptions surrounding the use of Data Illustrator and Data Animator. These assumptions are the "dirty little secrets" of our tools. The things that we often overlooked or outright ignored to make our research goals more feasible. In the following section I will reveal the assumptions that we made about the user, the datasets they hope to visualize, their creative task, the products they export, and the tools they use. These assumptions are inspired by a paper on critical reflections [157] that I co-authored with the teams from Lyra [23] and Charticulator [27]. I have refined and added to these assumptions and relate them to specific assumptions we made for Data Illustrator and Data Animator.

The User: Graphic Design Experience – By augmenting familiar design tools, Data Illustrator and Data Animator assume a certain level of familiarity with other interactive design applications. Data Illustrator assumes experience with vector graphic editors (e.g., Adobe Illustrator [19], Sketch [20]). While Data Animator leverages an understanding of keyframe animation in a timeline interface (e.g., Adobe After Effets [29]) and prototyping transitions between user interfaces or visual states (e.g., Adobe Xd [30], InVision [31], Principle [112]). We hypothesize that familiarity with design tools act as a catalyst for designers to learn our applications. However, it is unclear if experience with graphic design practices is the only factor at play. The participants from our studies have additional expe-

rience outside of graphic design. Perhaps experience with visualization systems, toolkits or grammars affects learnability for our tools.

On the other hand, similarity with graphic design practices could work against adoption by designers. Our approach is predicated on the assumption that designers are comfortable with computer-generated operators that require computational thinking to predict design outcomes. Computational thinking is an approach where the designer must express their goals in a manner so that a computer could also execute them [159]. At times the computer's model for a design is at odds with designers' mental model for constructing visualizations. I further reflect on how our approach necessitates computational thinking in Section 5.2.

The Data: Cleaned and Tidy – Our approach assumes that the data is an appropriately formatted CSV (comma-separated values) file. The data file is tidy: there are no missing or incorrect values, date fields are in expected format, and it has already been filtered. We also expect the data to be on the smaller side: hundreds of data rows instead of ten of thousands. Finally, the data is static, meaning that the structure or values of the data will not change at some later point in the authoring process. We speculate that these data assumptions would complicate the authoring process for designers. Given the data in a particular format, certain designs are impossible for designers to create without filtering or re-formatting the data. For designers this creates frustrating scenarios where it is unclear how to format the dataset to realize a desired visualization. Although Data Illustrator currently supports data nesting operations for binding multiple data tuples to a shape with repeat and partition, it does not support additional data transformations. Although data wrangling capabilities [160] are out of scope for these tools, future work should consider methods to assist designers in structuring or transforming datasets into compatible data schemas.

The Task: Visualization Design, Not Authoring – Our approach uses re-creation tasks to evaluate whether designers can understand and use the framework to compose visualiza-

tions. While the re-creation task is not a replica of the design process, it allows us to choose visualizations that cover all the concepts and features in our tool, and to compare participants' performance objectively. *Visualization authoring* is a more narrow task within the broader visualization design process. *Visualization design* is the general process of visualizing data. For the purpose of data analysis, a design task includes understanding the domain problem, the dataset, resulting insights, the visual representation, as well as how to author a design using our tools [161]. It is unclear how our approach holds up in organic design practice. For designers who are starting with a blank canvas, there is little guidance on how to proceed in the design process. Moreover, our approach lacks support for changing or considering alternative designs. Both tools restrict designers to a rigid creative experience. Future work should consider methods for designers to brainstorm ideas and consider design alternatives. For instance, how might these systems support re-usable components so that portions of a design can be re-used for an alternative direction. In Section 5.3, I address future research to understand and improve our approach for organic design practices.

The Goal: Prototype or Final Product? – In pursuit of creating a final visualization product, designers might produce versions at incremental levels of fidelity before realizing a final design. These intermediate visualizations (e.g., sketches, prototypes) provide feedback on the feasibility of a design idea without requiring the full effort and time required for a final product. To finalize a design, designers add refinements to the visualization (e.g., annotations, interactive components, visual embellishments) or devise constructs to embed them in an interactive web page (e.g., responsive web designs, navigation controls, page layout). This begs the question: what level of fidelity do Data Illustrator and Data Animator support? What is the intention for the exported products from these tools?

Both Data Illustrator and Data Animator are capable of exporting a visualization. In Data Illustrator, designers can export to Scalable Vector Graphics (SVG) to ensure high-fidelity display across a range of screen resolutions. However, exporting the visualization

permanently removes the backing dataset from the design. Once a designer makes refinements downstream, they cannot go back to change data-driven aspects of the graphics. Furthermore, designers require the ability to create visualizations that work on various screen sizes for modern browsing (e.g., mobile, tablets, desktop). A vector graphic image scales independent of screen resolution, however the aspect ratio is fixed on export. While other tools support updating the layout of a visualization based on the available chart dimensions [27], known as fluid design support, additional research should address responsive designs that re-flow charts based on available screen sizes [162]. Data Animator exports interactive web pages that contain a GPU-enhanced canvas for rendering animated transitions. While exporting a web page would suggest sharing a final product on the web, the page is an otherwise blank HTML template. Designers might want to further customize the page to include narrative refinements such as an annotation layer. Future work for Data Animator should support designers in adding these complementary elements to data stories.

So, what is the intended level of fidelity for these tools? The answer to that question remains an ambiguity to even me, their creator. We set out to implement tools capable of polished, ready-to-share visualizations, but achieving that level of polishing would likely incur system features outside of our research scope. My educated guess is that these tools support a level of fidelity somewhere between prototypes and final products – where exactly on that spectrum they reside depends on the user. Some designers with programming experience or advanced design skills consider Data Illustrator and Data Animator as prototyping tools to test out ideas or produce initial drafts to be refined in other tools. A less discerning designer might consider the supported output formats as final versions to share immediately. Our approach assumes it is up to the designer to decide what they plan to do with their visualization.

The Tools: A Closed Ecosystem – Our approach for Data Animator assumes a closed tool ecosystem where only static visualizations from Data Illustrator can be imported into Data Animator. This assumption can be limiting, as Data Illustrator is not necessarily integrated

in many designers' workflows. To author animated data graphics based on static visualizations, however, basic vector graphics formats such as SVG are not sufficient. It is essential to provide information about data binding, visual object grouping and hierarchy so that the system can automate the matching of objects between static graphics. To date, there has been no universal format that describes complete information about a static data visualization, and different authoring systems for animated data graphics are adopting different standards or formats. For example, Canis [99] requires a data-enriched variant of SVG to be used as input format. In future work could investigate methods to translate visualizations in other formats to the Data Illustrator framework. Such a tool may parse a vector graphic, analyze the properties and structure of visual objects, and ask designers to provide minimal annotations. The output of such a tool can be directly fed into Data Animator as input static visualizations. In Section 5 I elaborate on this approach for recognizing Data Illustrator designs from existing SVG examples.

5.2 Reflecting on the Lazy Data-Binding Approach

By deploying Data Illustrator on the web, we have learned a lot about our approach. Since the time of deployment, we have received feedback from users about the visualizations they created, requests for new features, and the occasional bug. This feedback provides insights into the advantages and disadvantages of our approach in real-world practices. By deploying Data Animator, we hope to gather similar user feedback.

Since Data Illustrator's introduction, the space for visualization authoring tools has matured as new research approaches arrive on the scene. Of note are approaches based on declarative grammars [23, 100] and constraint-based authoring tools [27]. Users can directly compare and contrast these tools, often times taking to social media to broadcast their preferences. Informed by such user feedback and my own understanding of the research space, I have reflected on the strengths and limitations of our lazy data-binding approach. In this approach we favor drawing vector graphics first then attaching data; data bindings

are applied when necessary, only constraining the property they are bound to. Furthermore, generative operators such as repeat and partition reduce the manual effort to copy shapes, attach data, and configure sophisticated layouts. These generative operators provide the information required by Data Animator to generate animate transitions through automatic matching of shapes between two static visualizations. Data Animator leverages the innate structure (e.g., collections, groups, peer objects, object, properties) to coordinate the temporal rhythm of animations with hierarchical keyframing. Below we reflect on the strengths and limitations of this approach.

Heuristic Ambiguity – The lazy data-binding approach reduces the time and effort it takes to encode visual properties based on data. In Data Illustrator, with a peer shape selected, designers only need to choose which data attribute to bind to a visual property in the inspector - the system takes care of the rest. Data Illustrator recognizes the appropriate scale to use based on the data type and visual property, transforms the data with that scale, and applies the resulting visual properties to the appropriate peer shapes. Within this model there are heuristic rules that govern: how data is mapped, how data generates shapes, and how to layout shapes. For example, when designers bind a numerical data attribute to "fill color" the tool heuristically produces a gradient scale for that mapping. However, the designer might desire a stepped or nominal color scale in this situation. More complicated issues arise when heuristic rules govern how shapes are repeated and attached to data. For example, designers cannot repeat the peer shapes within a collection to realize a small multiple design. In these scenarios, designers lack the agency to override this default behavior. A misalignment of heuristic rules with designers' expectations can be frustrating for designers and require them to find work around solutions.

Informed by this lesson, Data Animator provides capabilities to view the results of the matching algorithm and manually disambiguate those matches. For example, a designer connects a transition between two bar charts in Data Animator. The system will match the set of peer rectangles between those two vis boards and the resulting animation tweens the

differing height property between the two bar charts. This behavior makes sense, but this matching might also be undesirable to the designer. Perhaps the first bar chart represents an entirely different dataset then the second bar chart. To override this match, the designer can remove the match within the "Object Matcher" interface. The resulting animation allows the designers to specify preset effects (e.g., "Wipe In", "Wipe Out") to the two resulting sets of entering and exiting rectangles. Future lazy data-binding approaches should provide similar avenues to override heuristic rules. In the declarative grammar approach [69, 95, 89, 100], designers can specify partial to full specifications for a design, the visualization grammar fills in that partial specifications using defaults.

Direct Manipulation – The lazy data-binding approach seeks to emphasize direct manipulation of graphics in the canvas. We made this decision as an assumption that designers want to manually select and control graphics. In the case of Data Illustrator, designers can break layouts to specify their own freeform layout. Since data-bindings only act as constraints, designers can make changes to non data-bound properties such as the y-position of peer rectangles when their x-position is mapped to a data attribute. Designers can also gain immediate feedback on changes to visual mappings by editing axes and legends in the canvas. However, certain user actions require the use of panel interfaces such as creating data-bindings or a generative operation (e.g., repeat, partition). Despite our goal, Data Animator lacks support for direct manipulation in the canvas. As the canvas in the "Timeline Editor" only supports previewing animated transitions, without the ability to select and edit graphics. This decision was largely due to implementation constraints. Future improvements to Data Animator should provide such an ability to select shapes in the preview canvas or use direct manipulation to disambiguate object set matches within the matching interface. However, the assumption that designers prefer to use direct manipulation could be speculative. Design tools such as Adobe Illustrator [19] and Adobe After Effects [29] require the user of numerous panel interfaces. Also tools that heavily emphasize direct manipulation such as iVolver [24] and DataInk [26] lack capabilities to scale with larger

datasets.

Design Rigidity – Repeat and partition provide two generative operations that can be combined to realize nested layouts in Data Illustrator. The combination of repeat and partition provide a generative language to realize a variety of layouts. However, this requires authors to have a solid understanding of how these two operations work. Also, designers must commit to a layout when constructing it. Once the designer decides to break a layout, the shapes cannot be organized into a layout downstream of this design decision. For this reason, our approach lacks design flexibility to pivot to an alternative design. The only option is to start over again from scratch. In constraint-based layout approaches such as Charticulator [27] supports the specification of layouts (e.g., stack, grid, circle-packing) to layout an entire chart or sub-layouts. Layouts can be applied downstream of design decisions, because constraint-based solvers place shapes within the available chart dimensions.

Data Animator also suffers from design rigidity. For example, a designer creates and edits a staggering effect for an object set. If the designer manually removes the matching for that object set, they will lose their previously specified staggering design. Also, Data Animator requires designers to create groupings ((e.g., groups, collections) in Data Illustrator *a priori*. Designers cannot specify new groupings in Data Animator to take advantage of nested pacing techniques. Future implementations of Data Animator should provide the ability to specify groupings on the fly. I believe such design flexibility is a major consideration for more organic design processes that I discuss later in Section 5.3.

Two Composable Operators – Repeat and partition are the cornerstone of our approach for scaling lazy data binding. These two generative operators are relied on heavily throughout Data Illustrator and Data Animator. The strength in our approach is that there are only two operations for designers to understand and use. Rather than a number of different generative operators, data transformations, and layout constructs, designers are able to compose a variety of layouts and data mappings with these two functions. Feedback from design-

ers indicates this is a strength, as once they understand how these operations work, they are able to realize a variety of designs. However, this requires these two operations to function in different ways depending on the selected object (e.g., object type, number of objects). Improvements to Data Illustrator will need to build on top of these operations for expressivity without exasperating their simplicity.

Computational Thinking – Our approach aims to augment design tools to provide designers with a familiar experience for authoring visualizations. Despite the many design considerations to adhere to graphic designer practice, designers often comment that they are visually programming when using these tools. Visually programming, or computational thinking, is an approach where the designer must express their actions in ways that a computer could execute [159]. Data Illustrator requires computational thinking to anticipate how the repeat and partition operators will generate shapes based on the input data. Data Animator also requires designers to understand the resulting animation from the matching algorithm. Both tools try to alleviate computational thinking through exposing the computer's generative model to the designer at different points. For example, in Data Illustrator, as designers reveal peer shapes within a repeat grid, it is meant to help them build an understanding of this generative operation. Similarly, in Data Animator, the "Object Matcher" affords viewing which specific shapes have been matched by data field between the two static visualizations in a transition. Future empirical research needs to investigate cases where users do not adopt computational thinking, or if computer-assisted graphic applications can altogether avoid the trappings of computational thinking. To my knowledge, visualization authoring tools can only peel back the curtain to reveal their inner workings. In my opinion, this limited approach trumps obfuscating generative operations from the user entirely.

5.3 Supporting Organic Design Practices

As previously mentioned in Section 5.1, we assumed a visualization authoring task rather than a design task for these systems. *Visualization design* is the general process of visualizing data. Visualization design often includes the exploration of design possibilities in an iterative process; designers make decisions on visual encodings, data content, and the overall composition [8]. Understanding how well Data Illustrator and Data Animator support organic design practices helps to evaluate the feasibility of these tools. How well these tools support real-world design scenarios correlates to their adoption by the graphic design community. In this dissertation, we have evaluated Data Illustrator and Data Animator's effectiveness to support an authoring task with re-creation studies (Section 3.4.2 and 4.7.2, respectively). However, here, we discuss future work to investigate our approach's capacity to support a broader design process.

Possible methods to evaluate design tools under an organic design task is with longitudinal, freeform, and adoption case studies. In their in-depth study, Jacobs et al. commissioned an artist to create several pieces of art [108]. The study evaluated the system's performance in a realistic, creative practice. Similarly, Ren et al. identify adoption case studies as a method to investigate the creativity and expressivity supported by visualization authoring systems [34]. Adoption case studies involve partnering with one or more designers to observe their use of the system with their own data in their own environment. Free-form studies have been used to evaluate the creativity of visualization authoring systems in the past [34], typically these studies occur in laboratory or workshop settings under a limited time duration. Yet laboratory studies introduce controlled measures such as standardized computer setups or cleaned and pre-processed datasets. Future research should evaluate the creativity of Data Illustrator and Data Animator in realistic design environments with no restrictions on the user environment, input dataset, or target visualization products. In such a study, user-reported data such as journaling provides insights into the strengths and

limitations of these tools in-situ, as the design process progresses. Results of a longitudinal study could provide insights into how designers ideate and explore visualization designs with these tools. Such a study could also provide feedback on the expressive capabilities of these tools under realistic, non-curated conditions.

5.4 Expressive Visualization as a Service

My dissertation has focused on empowering a small population of highly skilled graphic designers. Despite our circumstantial proof that people without graphic design experience could use these tools, it is not plausible for a general audience to understand and effectively use Data Illustrator or Data Animator. Instead of watering-down these tools for a more general user, can we instead leverage the expressive visualizations created by designers for non-designers?

Non-designers might include business administrators, data analysts, customer experience strategists, bloggers, teachers – really anyone who wants to create their data. To empower this user, can we define expressive visualizations as re-usable components to be re-purposed for different datasets? In this approach designers would construct custom visualizations for their dataset based on expressive visualizations created by designers. Instead of choosing an entire design, as is the case in template editors [77, 14, 78, 15], users would construct their design with re-usable components.

There are two main challenges to supporting expressive visualizations as re-usable components:

- How can a framework support *custom or bespoke* data visualizations as re-usable components?
- How to support non-designers in constructing visualization designs that work for their dataset and presentation objectives?

Addressing the first challenge requires a robust framework. The framework would need

to support the specification of re-usable visualization design components. Those re-usable components would need to be sourced from a large collection of existing visualization designs. The collection of visualization designs in the Data Illustrator and Data Animator galleries [163, 164] are not a large enough sample size. Instead, future research could use existing collections from D^3 visualizations on the web [165]. Visualizations implemented in SVG and programmed with D^3can be deconstructed [166, 158] as reusable style templates. However, additional research would be needed to generate re-usable components (e.g., glyphs, layouts, axis or legend styles) from a given SVG visualization design. This approach is complicated by visualization designs that include domain specific information, or trends that are unique to the original dataset. Re-usable components would need to be generalize-able. The framework would combine these components to compose all-together different visualizations from the original. In this way, non-designers would not just be using visualization templates, but making their own (albeit constrained) design decisions.

The second challenge requires formative research into how non-designers approach constructing expressive visualizations from re-usable components. Recent research systems recommend visualizations to users based on selected data attributes [81, 82], or paired with data insights and facts[167]. For users without an understanding of their data, these recommendations help to choose appropriate visualizations for their data schema. However, these systems recommend visualizations based on a declarative grammar [89], whereas recommendations for re-usable components is an unexplored research direction.

5.5 Authoring Interactive Visualizations

According to Shneiderman, interaction is a crucial first step for visual-information seeking as it provides opportunities for users to "overview first, zoom and filter, then [provide] details on demand" [168]. Yi et al. propose seven general categories of interaction techniques from the perspective of user intent (e.g., select, explore) [169]. Mantras and taxonomies from previous research indicate a better understanding of interaction for exploratory than

explanatory visualization. Only until recently have researchers investigated the effectiveness and purpose of interaction in data storytelling [51, 170]. Practitioners have cast doubt on traditional visualization interaction as a useful feature in storytelling [45]. Gregor Aisch and Archie Tse from the *New York Times* recently commented that readers are unlikely to interactively explore visualizations. Visualization designers should instead focus their efforts on revealing important content through passive forms of interaction such as narrative navigation [46, 47]. Aisch has since clarified that "interactive graphics are still great," claiming that interaction builds trust in the data and allows users to gain further details [48]. Future work on Data Animator should include support for "detail on demand" interactions such as highlighting and filtering within a visualization. Supporting more involved interactions such as updating data for a graphic based on user input requires an authoring system to generate new visualizations on the fly. Recent work by Zong et al. introduces "interaction design by specification" to the Lyra system, allowing designers to program by example [171]. However, these interactions are limited to a set of interactions afforded by the underlying Vega grammar [95, 89]. Building on this work, researchers should consider methods for designers to dynamically author interactions via visual metaphors.

Research opportunities also exist for authoring novel interaction techniques that go beyond traditional, exploratory interactions. For example, in the "You Draw It" series of visualizations from the *New York Times*, readers are prompted to draw their guess for a temporal trend [172, 173, 174]. After the reader's guess the article responds with the actual trendline, allowing the reader to see if they overestimated or underestimated. Kim et al. found evidence that prompting readers to draw trends is an effective method to improve recall and encourages engagement [175, 176]. Hohman et al. survey interactive articles from journalism and digital media showcases [170]. Their results demonstrate how interactive articles can help boost learning and engagement for readers compared to static alternatives. Despite the expository power of these interactions, similar barriers exist for graphic designers to create such interactive articles. Designers must learn how to program to create novel

interactive and dynamic media. Future research should seek to empower non-programmers to showcase their creativity in interaction methods for data storytelling.

5.6 Re-Imagining Data Storytelling

Data storytelling is traditionally composed of a linear sequence of visualizations with connecting animated transitions. Segel and Heer classify the order of a story as reader-directed paths and linear (author-directed) [4]. Linear storytelling refers to the author directing the narrative. For example, the animated data graphics created with Data Animator follow a linear order. This format allows the author to control the experience of the reader, helping to build up a complex explanation from simple components in a story. Linear-storytelling supports a traditional plot-driven account of data insights. Reader-driven narratives refer to non-prescribed ordering, minimal messaging, and high interactivity. Similar to visual analysis, the reader interacts laterally – filtering, highlighting, and updating the data within the same visualization. This format often lacks a strong story, pushing narrative agency toward the reader.

Novel narrative formats are beginning to emerge that combine these two approaches. Non-linear storytelling balances narrative agency between author and reader. Giorgia Lupi describes non-linear storytelling as a "design method, based upon [the author] layering multiple sub-narratives over a main construct" to realize a spatial build-up of visual narrative paths to explore [44]. Non-linear storytelling affords the reader agency to explore tangential stories within a larger visualization. Non-linear stories differ from reader-directed, because the author curates tangential visual stories as options for the reader. This format lends itself well to engaging readers with personalized data [177]. For example, *the New York Times'* running COVID-19 article provides a format for readers to engage with data about their country, their state, and their local county [178]. The article "Your Personal Carbon History" from *the Pudding* is another example of non-linear storytelling [179]. The dynamic article prompts the reader to share their birth year and curates a personal-

ized narrative about carbon emissions relative to the reader's lifetime. Creating non-linear narratives requires time, effort, and resources to curate multiple tangential tales or adapt stories to user input. Future research directions should provide tools for non-linear storytelling that reduce the author's burden. For example, future research could assist authors by recommending components to a data story. Additional research could assist with recommending intermediate visualizations to improve apprehension between transitions. For example, such a system could recommend intermediate visualizations that minimize visual changes and divide complex changes between animated transitions [54].

CHAPTER 6

CONCLUSION

This dissertation explores opportunities to research, design, and implement data visualization authoring tools for graphic designers. Guided by three research questions (RQ1, RQ2, RQ3), our approach augments graphic design practices by introducing familiar user interfaces and interactions on top of expressively powerful data graphic frameworks (RQ1–3 are summarized in Table 6.1). My overarching research goal is to broaden the practice and participation in data visualization to graphic designers. This dissertation takes a step toward reaching that goal by introducing two interactive systems: Data Illustrator and Data Animator. These two systems provide graphic designers with an understandable, feasible, and effective method to author expressive data visualizations (without writing textual code). I evaluated the usability of these systems with re-creation studies and demonstrate their expressive power through example galleries.

Table 6.1: A summary of the research questions that guided this dissertation. Contributions address each RQ for authoring *static* and *animated* visualizations.

	Research Question	**Static**	**Animated**
RQ1	What are the building blocks to a static and animated visualization framework that graphic designers can understand and employ?	[116, 35]	[36]
RQ2	Leveraging this framework, is it feasible to design and develop authoring systems that support designers to author expressive visualizations (without writing textual code) and fit within the tradition of graphic designers?	[35]	[37]
RQ3	Can graphic designers understand and effectively use these tools to author expressive visualizations?	[35]	[37]

In Chapter 3, I presented Data Illustrator: this tool augments vector design tools, allowing designers to draw first and introduce data as needed. We contribute two generative

operators, repeat and partition, which minimize the effort needed to copy shapes and attach data. This provides a visual language for specifying nested layouts by data. Lazy data bindings increase authoring flexibility, as data bindings only act as constraints on the data bound property of each shape. Through these contributions, Data Illustrator scales the lazy data binding approach, as it exhibits expressivity comparable to declarative grammars and programming toolkits.

In Chapter 4, I introduced Data Animator as an extension of Data Illustrator for authoring animated data graphics. Data Animator augments familiar motion graphic and prototyping tools by adopting a keyframe animation approach. This system leverages the Data Illustrator framework to automatically match sets of objects between two static visualizations, allowing designers to rapidly generate animated transitions by default. Data Animator allows designers to view and disambiguate generated matches in a novel matching interface. Finally, we introduce hierarchical keyframing, which allows designers to combine pacing techniques (e.g., staging, staggering) in a timeline editor. With Data Animator, designers are able to coordinate the temporal rhythm of an animation.

In Chapter 5, I reflected on the assumptions we made during the design, implementation, and deployment of these systems. I also discussed the strengths and limitations of the lazy data binding approach, contrasted against related visual builder systems. Finally, inspired by the contributions of this dissertation, I proposed future research directions in the vein of communicative visualization.

Appendices

APPENDIX A

SURVEY OF STATIC DATA GRAPHICS

Table A.1: A survey of example static data graphics from online sources.

Title (URL)	**Authors or Publication**	**Year**
A Game of Two Halves	Dave Gardiner	2014
Aftermarket Education	Beutler Ink	2014
Agentschappen in beeld 2013	Roel de Jonge	2014
An Actor's Life	The Slow Journalism Company	2014
Bay Area Bike Share Station Activity	Anastasia Papadi, Juan Francisco Saldarriaga	2014
Breathing City	Joey Cherdarchuk	2014
Creative Routines	RJ Andrews	2014
Data Enrichment Visualization	WizArts Inc	2014
EU Humanitarian Aid	The Visual Agency	2014
Expo 2015: How Milan is Connected to the World	The Visual Agency	2014
Game of Thrones Transit Map	Michael Tyznik	2014
Global Trends Challenging Cities	Rambler and Co	2014
Green Nudge Energy Visualization	WizArts Inc	2014
In orbit but hardly alone	Thibaud Tissot, Raphael Schön	2014
Lifelines	Oliver Uberti	2014
Movimento Pendular	Dimitre Lima Ana, Paula Megda	2014
News Stream	Manuel Reitz	2014
Population Lines	James Cheshire	2014
Press	Gaia Russo	2014
Rappers, Sorted by Size of Vocabulary	Matthew Daniels	2014
Sabinal	Alberto Lara	2014
Seattle Seahawks' Franchise History	Chartball	2014

Title (URL)	Authors or Publication	Year
Starmap of Nobel Prize	Caixin Media Company	2014
The Analytical Tourism Map of Piedmont	Marco Bernardi, Federica Fragapane, Francesco Majno	2014
The Black Data of Piemonte	Sara Piccolomini	2014
The Depth of the Problem	The Washington Post	2014
Top 100 Companies Worldwide	The Visual Agency	2014
Tumourmonger	Tobias Sturt	2014
Visualize Pi Noise	Ellie Balk, Nathan Affield	2014
Voting and Attendance in Slovenian Parliament	Marko Plahuta	2014
Weather Radials	Timm Kekeritz	2014
What is Wikipedia about?	Paul-Antoine Chevalier, Arnaud Picandet	2014
What Teachers Think	The Visual Agency	2014
Wikiflows - One Year on Wikipedia	Valerio Pellegrini, Michele Mauri	2014
Wiring the World	National Geographic Society	2014
World Cup 2014 Wall Chart	Leigh Riley	2014
Worldwide Holiday Costs Barometer 2014	Andrew Park	2014

APPENDIX B

SURVEY OF ANIMATED DATA GRAPHICS

Table B.1: A survey of example animated data graphics from online sources. Each example includes at least one animation instance that directly alters or incorporates the data graphic.

Title (URL)	**Authors or Publication**	**Year**
342,00 Swings Later, Derek Jeter Calls it a Career	The New York Times	2014
512 Paths to the White House	The New York Times	2012
A 3-D View of a Chart That Predicts The Economic Future: The Yield Curve	The New York Times	2015
A Breathing Earth	Nadieh Bremer	2016
A History of Sumo	Five Thirty Eight	2016
A Visual Introduction to Machine Learning - Part I	Stephanie Yee, Tony Chu	2015
A Visual Introduction to Machine Learning - Part II	Stephanie Yee, Tony Chu	2018
Ali Wong - Structure of Stand Up Comedy	The Pudding	2018
An Interactive Visualization of Every Line in Hamilton	The Pudding	2017
Beautiful in English	Nadieh Bremer	2016
Beauty Brawl: How diverse are makeup shades?	The Pudding	2018
Berlin-Marathon 2016	Berliner Morgenpost	2016
Bubble to Bust to Recovery	Bloomberg	2014
Bussed Out	The Guardian	2017
Can we talk about the gender pay gap?	Washington Post	2017
Cracking the Mystery of Egg Shape	Science Magazine	2017
Craft beer - so hot right now	The Pudding	2017
Crowd Sourcing the Definition of Punk	The Pudding	2017
Explore Adventure	Shirley Wu	2016
Extensive Data Shows Punishing Reach of Racism for Black Boys	The New York Times	2018

Title (URL)	Authors or Publication	Year
Film Money: A data story	Lars Verspohl	2016
Free Willy and Flipper by the Numbers	The Pudding	2017
Gender Gap at the Olympics	Wall Street Journal	2018
How Americans Die	Bloomberg	2014
How many high school stars make it in the NBA?	The Pudding	2019
How the Recession Reshaped the Economy, in 255 Charts	The New York Times	2014
How the U.S. and OPEC Drive Oil Prices	The New York Times	2015
I'm not feeling well	Google News Lab	2017
Income and Financial Stability in America (50/30/20)	Amy Cesal, Daniel Doherty	2016
Most Common Occupation by Age	Nathan Yau	2018
Oil Spilled at Sea	Reuters	2018
Out of Sight, Out of Mind	Pitch Interactive	2015
Royal Constellations	Nadieh Bremer	2016
Scientific Proof that Americans are Completely Addicted to Trucks	Bloomberg	2015
Swimming World Records throughout History	Irene de la Torre Arenas	2017
Table for one	The Pudding	2017
The Fallen of World War II	Neil Halloran	2015
The Four Days in 1968 that Reshaped D.C.	Washington Post	2018
The Rhythm of Food	Google News Lab	2018
The Shadow Peace	Neil Halloran	2016
The Shape of Slavery	The Pudding	2017
The Timing of Baby Making	The Pudding	2017
The Unlikely Odds of Making it Big	The Pudding	2017
This is every active satellite orbiting earth	Quartz	2015
Top 15 Best Global Brands Ranking	The Rankings	2019
Trump's Year in Tweets	Gramener	2017
Twenty Years of the NBA Redrafted	The Pudding	2017
U.S. Gun Deaths	Periscopic	2018

Title (URL)	Authors or Publication	Year
Waiting for a Table	Nathan Yau	2018
Wealth Inequality in America	Politizane	2012
What was germany searching for?	Moritz Stefaner	2017
What's it like to get trolled all day long?	Hindustan Times	2017

REFERENCES

[1] E. Meeks and S. Lu, *The 7 kinds of data visualization people*, https://medium.com/visualizing-the-field/the-7-kinds-of-data-visualization-people-9964e80443a7, 2017.

[2] E. Meeks, *3rd Wave Data Visualization*, https://towardsdatascience.com/3rd-wave-data-visualization-824c5dc84967, 2018.

[3] Kantar, *Information is Beautiful Awards*, https://www.informationisbeautifulawards.com/, 2017.

[4] E. Segel and J. Heer, "Narrative visualization: Telling stories with data," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 16, no. 6, pp. 1139–1148, 2010, doi:=10.1109/TVCG.2010.179. DOI: 10.1109/TVCG.2010.179.

[5] B. Lee, N. H. Riche, P. Isenberg, and S. Carpendale, "More Than Telling a Story: Transforming Data into Visually Shared Stories," *IEEE Computer Graphics and Applications*, vol. 35, no. 5, pp. 84–90, Sep. 2015. DOI: 10.1109/MCG.2015.99.

[6] C. D. Stolper, B. Lee, N. Henry Riche, and J. Stasko, "Data-Driven Storytelling Techniques: Analysis of a Curated Collection of Visual Stories," in *Data-Driven Storytelling*, N. Henry Riche, C. Hurter, N. Diakopoulos, and S. Carpendale, Eds., A K Peters/CRC Press, 2018.

[7] N. H. Riche, C. Hurter, N. Diakopoulos, and S. Carpendale, *Data-Driven Storytelling*. CRC Press, 2018.

[8] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer, "Reflections on how designers design with data," in *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, ser. AVI '14, Como, Italy: ACM, 2014, pp. 17–24. DOI: 10.1145/2598153.2598175.

[9] A. Cairo, *The Functional Art: An Introduction to Information Graphics and Visualization*. New Riders, 2012.

[10] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 17, no. 12, pp. 2301–2309, 2011. DOI: 10.1109/TVCG.2011.185.

[11] H. Wickham, "ggplot: An Implementation of the Grammar of Graphics," *R package version 0.4. 0*, 2006.

[12] C. Reas and B. Fry, "Processing: programming for the media arts," *AI & Society*, vol. 20, no. 4, pp. 526–538, Sep. 2006. DOI: 10.1007/s00146-006-0050-9.

[13] G. G. Méndez, U. Hinrichs, and M. A. Nacenta, "Bottom-up vs. top-down: Trade-offs in efficiency, understanding, freedom and creativity with infovis tools," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '17, Denver, Colorado, USA: ACM, 2017, pp. 841–852. DOI: 10.1145/3025453.3025942.

[14] Kiln Enterprises Ltd, *Flourish*, https://flourish.studio, 2020.

[15] Datawrapper GmbH, *Datawrapper*, urlhttps://www.datawrapper.de/, 2020.

[16] *Plotly*, https://plot.ly.

[17] M. Mauri, T. Elli, G. Caviglia, G. Uboldi, and M. Azzi, "RAWGraphs: A visualisation platform to create open outputs," in *Proceedings of the ACM Italian CHI Conference*, 2017, 28:1–28:5. DOI: 10.1145/3125571.3125585.

[18] Tableau Software, *Tableau Software: Business Intelligence and Analytics*, https://www.tableau.com/, 2020.

[19] Adobe Systems Inc., *Adobe Illustrator CC*, http://www.adobe.com/products/illustrator.html, 2020.

[20] Bohemian Coding, *Sketch - the digital design toolkit*, https://www.sketchapp.com/, 2020.

[21] B. Victor, *Drawing dynamic visualizations*, http://vimeo.com/66085662, 2013.

[22] D. Ren, T. Höllerer, and X. Yuan, "iVisDesigner: Expressive interactive design of information visualizations," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 20, no. 12, pp. 2092–2101, 2014. DOI: 10.1109/TVCG.2014.2346291.

[23] A. Satyanarayan and J. Heer, "Lyra: An interactive visualization design environment," *Computer Graphics Forum (Proceedings of EuroVis)*, vol. 33, no. 3, 2014. DOI: 10.1111/cgf.12391.

[24] G. G. Méndez, M. A. Nacenta, and S. Vandenheste, "iVoLVER: Interactive visual language for visualization extraction and reconstruction," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2016, pp. 4073–4085. DOI: 10.1145/2858036.2858435.

[25] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic, and H. Pfister, "Data-driven guides: Supporting expressive design for information graphics," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 23, no. 1, pp. 491–500, 2017. DOI: 10.1109/TVCG.2016.2598620.

[26] H. Xia, N. Henry Riche, F. Chevalier, B. De Araujo, and D. Wigdor, "DataInk," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '18, ACM Press, 2018, pp. 1–13. DOI: 10.1145/3173574.3173797.

[27] D. Ren, B. Lee, and M. Brehmer, "Charticulator: Interactive Construction of Bespoke Chart Layouts," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 789–799, Jan. 2019. DOI: 10.1109/TVCG.2018.2865158.

[28] E. Meeks, *D3 is not a Data Visualization Library*, https://medium.com/@Elijah_Meeks/d3-is-not-a-data-visualization-library-67ba549e8520, 2018.

[29] Adobe Systems Inc., *Adobe After Effects CC*, http://www.adobe.com/products/aftereffects.html, 2020.

[30] *Adobe Xd*, http://www.adobe.com/products/xd.html, 2020.

[31] InvisionApp Inc., *Invision*, https://www.invisionapp.com/, 2020.

[32] M. Brehmer, B. Lee, N. Henry Riche, D. Tittsworth, K. Lytvynets, D. Edge, and C. White, "Timeline storyteller: The design & deployment of an interactive authoring tool for expressive timeline narratives," in *Proceedings of the the Computation + Journalism Symposium*, 2019.

[33] F. Amini, N. H. Riche, B. Lee, A. Monroy-Hernandez, and P. Irani, "Authoring data-driven videos with dataclips," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 23, no. 1, pp. 501–510, 2017. DOI: 10.1109/TVCG.2016.2598647.

[34] D. Ren, B. Lee, M. Brehmer, and N. H. Riche, "Reflecting on the Evaluation of Visualization Authoring Systems : Position Paper," in *2018 IEEE Evaluation and Beyond - Methodological Approaches for Visualization (BELIV)*, IEEE, Oct. 2018, pp. 86–92. DOI: 10.1109/BELIV.2018.8634297.

[35] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko, "Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '18, Montreal QC, Canada: ACM, 2018, 123:1–123:13. DOI: 10.1145/3173574.3173697.

[36] J. Thompson, Z. Liu, W. Li, and J. Stasko, "Understanding the Design Space and Authoring Paradigms for Animated Data Graphics," in *Computer Graphics Forum*, Wiley Online Library, vol. 39, 2020, pp. 207–218. DOI: 10.1111/cgf.13974.

[37] J. Thompson, Z. Liu, and J. Stasko, "Data animator: Authoring expressive animated data graphics," in *Under review at the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '21, Japan, 2021.

[38] V. Chan, *Getting it right: why infographics are not the same as data visualizations*, 2017.

[39] P. Taei, *What Is an Infographic? And How Is it Different from a Data Visualization?* 2018.

[40] M. A. Borkin, A. A. Vo, Z. Bylinskii, P. Isola, S. Sunkavalli, A. Oliva, and H. Pfister, "What makes a visualization memorable?" *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2306–2315, Dec. 2013. DOI: 10.1109/TVCG.2013.234.

[41] M. A. Borkin, Z. Bylinskii, N. W. Kim, C. M. Bainbridge, C. S. Yeh, D. Borkin, H. Pfister, and A. Oliva, "Beyond memorability: Visualization recognition and recall," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 22, no. 1, pp. 519–528, 2016. DOI: 10.1109/TVCG.2015.2467732.

[42] L. Byrne, D. Angus, and J. Wiles, "Acquired codes of meaning in data visualization and infographics: Beyond perceptual primitives," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 509–518, Jan. 2016. DOI: 10.1109/TVCG.2015.2467321.

[43] D. Skau, L. Harrison, and R. Kosara, "An evaluation of the impact of visual embellishments in bar charts," *Computer Graphics Forum*, vol. 34, no. 3, pp. 221–230, 2015. DOI: 10.1111/cgf.12634.

[44] G. Lupi, *The Architecture of a Data Visualization*, https://medium.com/accurat-studio/the-architecture-of-a-data-visualization-470b807799b4, 2015.

[45] D. Baur, *The death of interactive infographics?* https://medium.com/@dominikus/the-end-of-interactive-visualizations-52c585dcafcb, 2017.

[46] G. Aisch, *Gregor Aisch - Information+ Conference*, https://vimeo.com/182590214, 2017.

[47] G. Tse, *Why We Are Doing Fewer Interactives*, https://github.com/archietse/malofiej-2016/blob/master/tse-malofiej-2016-slides.pdf, 2016.

[48] G. Aisch, *In Defense of Interactive Graphics*, https://www.vis4.net/blog/2017/03/in-defense-of-interactive-graphics/, 2017.

[49] J. Hullman and N. Diakopoulos, "Visualization rhetoric: Framing effects in narrative visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2231–2240, Dec. 2011. DOI: 10.1109/TVCG.2011.255.

[50] J. Hullman, S. Drucker, N. Henry Riche, B. Lee, D. Fisher, and E. Adar, "A deeper understanding of sequence in narrative visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2406–2415, Dec. 2013. DOI: 10.1109/TVCG.2013.119.

[51] S. McKenna, N. Henry Riche, B. Lee, J. Boy, and M. Meyer, "Visual Narrative Flow: Exploring Factors Shaping Data Visualization Story Reading Experiences," *Computer Graphics Forum*, vol. 36, no. 3, pp. 377–387, Jun. 2017. DOI: 10.1111/cgf.13195.

[52] M. Stefaner, *Some things I learned about data-driven storytelling in Schloss Dagstuhl*, https://medium.com/data-driven-storytelling/some-things-i-learned-about-data-driven-story-telling-in-schlo{\OT1\ss}-dagstuhl-b5ecfaef0910, 2016.

[53] D. Fisher, "Animation for Visualization: Opportunities and Drawbacks," in, https://www.microsoft.com/en-us/research/publication/animation-for-visualization-opportunities-and-drawbacks/, O'Reilly Media, Apr. 2010, ISBN: 9781449379865.

[54] J. Heer and G. Robertson, "Animated transitions in statistical data graphics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1240–1247, Nov. 2007. DOI: 10.1109/TVCG.2007.70539.

[55] Y. Kim, M. Correll, and J. Heer, "Designing Animated Transitions to Convey Aggregate Operations," *Computer Graphics Forum*, vol. 38, no. 3, pp. 541–551, 2019. DOI: 10.1111/cgf.13709.

[56] F. Chevalier, P. Dragicevic, and S. Franconeri, "The Not-so-Staggering Effect of Staggered Animated Transitions on Visual Tracking," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2241–2250, 2014. DOI: 10.1109/TVCG.2014.2346424.

[57] F. Du, N. Cao, J. Zhao, and Y.-R. Lin, "Trajectory bundling for animated transitions," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '15, doi:=10.1145/2702123.2702476, 2015, pp. 289–298. DOI: 10.1145/2702123.2702476.

[58] P. Dragicevic, A. Bezerianos, W. Javed, N. Elmqvist, and J.-D. Fekete, "Temporal distortion for animated transitions," in *Proceedings of the ACM Conference on*

Human Factors in Computing Systems (CHI), ser. CHI '11, New York, New York, USA: ACM Press, 2011, p. 2009. DOI: 10.1145/1978942.1979233.

[59] B. Tversky, J. B. Morrison, and M. Betrancourt, "Animation: can it facilitate?" *Int. J. Human-Computer Studies*, vol. 57, pp. 247–262, 2002. DOI: 10.1006/ijhc.1017.

[60] L. C. Rost, "One Chart, Nine Tools - Revisited," 2018, https://lisacharlotterost.de/datavistools-revisited.

[61] H. Rosling, *Gapminder*, https://www.gapminder.org/tools/, 2015.

[62] L. Grammel, C. Bennett, M. Tory, and M.-A. Storey, "A survey of visualization construction user interfaces," *EuroVis-Short Papers*, pp. 19–23, 2013.

[63] L. Wilkinson, *The Grammar of Graphics*. Springer-Verlag, 1999.

[64] H. Wickham, "A layered grammar of graphics," *Journal of Computational and Graphical Statistics*, vol. 19, no. 1, pp. 3–28, 2010. DOI: 10.1198/jcgs.2009.07098.

[65] ——, "ggplot2: Elegant graphics for data analysis," 2009.

[66] J. Heer and M. Bostock, "Declarative language design for interactive visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1149–1156, Dec. 2010. DOI: 10.1109/TVCG.2010.144.

[67] J. Heer, S. Card, and J. Landay, "Prefuse: A toolkit for interactive information visualization," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '05, ACM, 2005, pp. 421–430. DOI: 10.1145/1054972.1055031.

[68] M. Bostock and J. Heer, "Protovis: A graphical toolkit for visualization," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 15, no. 6, pp. 1121–1128, 2009. DOI: 10.1109/TVCG.2009.174.

[69] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-Lite: A grammar of interactive graphics," *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, vol. 23, no. 1, pp. 341–350, 2017. DOI: 10.1109/TVCG.2016.2599030.

[70] F. B. Viegas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon, "ManyEyes: A site for visualization at internet scale," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 13, no. 6, 2007. DOI: 10.1109/TVCG.2007.70577.

[71] *Infogram*, https://infogram.com.

[72] *Piktochart*, https://piktochart.com/.

[73] *Easel.ly*, https://easel.ly/.

[74] Y. Wang, H. Zhang, H. Huang, X. Chen, Q. Yin, Z. Hou, D. Zhang, Q. Luo, and H. Qu, "InfoNice," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*, ACM Press, 2018, pp. 1–12. DOI: 10.1145/3173574.3173909.

[75] *datamatic.io*, https://datamatic.io/.

[76] *Quadrigram*, http://www.quadrigram.com/.

[77] Microsoft, *Microsoft Excel*, https://www.microsoft.com/en-us/microsoft-365/excel, 2020.

[78] ——, *Power BI*, https://powerbi.microsoft.com/en-us/, 2020.

[79] Google, *Google Data Studio*, https://datastudio.google.com/, 2020.

[80] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: A system for query, analysis, and visualization of multidimensional relational databases," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 52–65, 2002. DOI: 10.1109/2945.981851.

[81] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer, "Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 649–658, Jan. 2016. DOI: 10.1109/TVCG.2015.2467191.

[82] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer, "Voyager 2: Augmenting visual analysis with partial view specifications," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '17, doi:= 10.1145/3025453.3025768, Denver, Colorado, USA: ACM, 2017, pp. 2648–2659. DOI: 10.1145/3025453.3025768.

[83] S. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein, "Interactive graphic design using automatic presentation knowledge," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '94, ACM, 1994, pp. 112–117. DOI: 10.1145/191666.191719.

[84] B. Lee, R. H. Kazi, and G. Smith, "Sketchstory: Telling more engaging stories with data through freeform sketching," *IEEE Transactions on Visualization and Computer Graphics*, VIS '13, vol. 19, no. 12, pp. 2416–2425, Dec. 2013. DOI: 10.1109/TVCG.2013.191.

[85] B. Lee, G. Smith, N. H. Riche, A. Karlson, and S. Carpendale, "Sketchinsight: Natural data exploration on interactive whiteboards leveraging pen and touch interaction," in *Visualization Symposium (PacificVis), 2015 IEEE Pacific*, IEEE, 2015, pp. 199–206. DOI: 10.1109/PACIFICVIS.2015.7156378.

[86] T. Schachman, *Apparatus: A hybrid graphics editor and programming environment for creating interactive diagrams*, http://aprt.us/, Dec. 2015.

[87] B. Myers, J. Goldstein, and M. Goldberg, "Creating charts by demonstration," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '94, ACM, 1994, pp. 106–111. DOI: 10.1145/191666.191715.

[88] R. Vuillemot and J. Boy, "Structuring visualization mock-ups at the graphical level by dividing the display space," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 424–434, 2018. DOI: 10.1109/TVCG.2017.2743998.

[89] Interactive Data Lab, *Vega: A Visualization Grammar*, https://vega.github.io/vega/, Aug. 2017.

[90] A. Satyanarayan, K. Wongsuphasawat, and J. Heer, "Declarative interaction design for data visualization," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, 2014, pp. 669–678. DOI: 10.1145/2642918.2647360.

[91] F. Amini, N. Henry Riche, B. Lee, C. Hurter, and P. Irani, "Understanding data videos: Looking at narrative visualization through the cinematography lens," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2015, pp. 1459–1468. DOI: 10.1145/2702123.2702431.

[92] A. Chalbi, "Understanding and designing animations in the user interfaces," https://hal.archives-ouvertes.fr/tel-01881889, PhD thesis, Université de Lille, 2018.

[93] F. Chevalier, N. H. Riche, C. Plaisant, A. Chalbi, and C. Hurter, "Animations 25 years later: New roles and opportunities," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*, ser. AVI '16, Bari, Italy: ACM, 2016, pp. 280–287. DOI: 10.1145/2909132.2909255.

[94] R. Cabello, *three.js – JavaScript 3D library*, http://threejs.org/, 2020.

[95] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer, "Reactive vega: A streaming dataflow architecture for declarative interactive visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 659–668, 2016. DOI: 10.1109/TVCG.2015.2467091.

[96] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 341–350, 2017. DOI: 10.1109/TVCG.2016.2599030.

[97] D. Ren, B. Lee, and T. Höllerer, "Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering," *Computer Graphics Forum*, vol. 36, no. 3, pp. 179–188, Jun. 2017. DOI: 10.1111/cgf.13178.

[98] T. Lin Pedersen and D. Robinson, *A Grammar of Animated Graphics: gganimate*, http://gganimate.com/, 2020.

[99] T. Ge, Y. Zhao, B. Lee, D. Ren, B. Chen, and Y. Wang, "Canis: A high-level language for data-driven chart animations," *Computer Graphics Forum*, vol. 39, no. 3, pp. 607–617, 2020. DOI: 10.1111/cgf.14005.

[100] Y. Kim and J. Heer, "Gemini: A grammar and recommender system for animated transitions in statistical graphics," *To appear in IEEE Transactions on Visualization and Computer Graphics*, 2020.

[101] Google News Lab, *Visualizing Google data*, https://trends.google.com/trends/story/US_cu_6fXtAFIBAABWdM_en, 2019.

[102] Microsoft, *Microsoft Powerpoint*, https://products.office.com/en-us/powerpoint, 2020.

[103] M. Brehmer, B. Lee, B. Bach, N. H. Riche, and T. Munzner, "Timelines revisited: A design space and considerations for expressive storytelling," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 23, no. 9, pp. 2151–2164, 2017. DOI: 10.1109/TVCG.2016.2614803.

[104] C. D. Stolper, M. Kahng, Z. Lin, F. Foerster, A. Goel, J. Stasko, and D. H. Chau, "Glo-stix: Graph-level operations for specifying techniques and interactive exploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2320–2328, 2014. DOI: 10.1109/TVCG.2014.2346444.

[105] G. G. Méndez, M. A. Nacenta, and U. Hinrichs, "Considering Agency and Data Granularity in the Design of Visualization Tools," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '18, ACM Press, 2018, pp. 1–14. DOI: 10.1145/3173574.3174212.

[106] J. Walny, S. Huron, and S. Carpendale, "An exploratory study of data sketching for visual representation," *Computer Graphics Forum*, vol. 34, no. 3, pp. 231–240, 2015. DOI: 10.1111/cgf.12635.

[107] J. C. Roberts, C. Headleand, and P. D. Ritsos, "Sketching designs using the five design-sheet methodology," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 419–428, Jan. 2016. DOI: 10.1109/TVCG.2015.2467271.

[108] J. Jacobs, S. Gogia, R. Měch, and J. R. Brandt, "Supporting expressive procedural art creation through direct manipulation," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '17, ACM, 2017, pp. 6330–6341. DOI: 10.1145/3025453.3025927.

[109] Tumult Inc., *Tumult hype 3.6*, https://tumult.com/hype/, 2019.

[110] A. Katz, *Layer Repeater - aescripts*, https://aescripts.com/layer-repeater/, 2020.

[111] MW Motion, *Blend - aescripts*, https://aescripts.com/blend/, 2020.

[112] Principle Inc., *Principle*, https://principleformac.com/, 2020.

[113] Apple Inc., *Keynote*, https://www.apple.com/keynote/, 2020.

[114] R. H. Kazi, F. Chevalier, T. Grossman, S. Zhao, and G. Fitzmaurice, "Draco: Bringing Life to Illustrations with Kinetic Textures," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14, Toronto, Ontario, Canada: ACM, 2014, pp. 351–360. DOI: 10.1145/2556288.2556987.

[115] R. H. Kazi, F. Chevalier, T. Grossman, and G. Fitzmaurice, "Kitty: Sketching Dynamic and Interactive Illustrations," in *Proceedings of the ACM Symposium on User Interface Software and Technology*, ser. UIST '14, Honolulu, Hawaii, USA: ACM, 2014, pp. 395–405. DOI: 10.1145/2642918.2647375.

[116] J. Thompson and J. Stasko, "Understanding data-driven visual encodings through deconstruction," *Poster at IEEE VIS 2016*, 2016.

[117] N. Holmes, *Designer's guide to creating charts & diagrams*. Watson-Guptill, 1984.

[118] P. B. Meggs and A. W. Purvis, *Meggs' History of Graphic Design*. John Wiley & Sons, 2016.

[119] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer, "Iterating between Tools to Create and Edit Visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 481–490, Jan. 2017. DOI: 10.1109/TVCG.2016.2598609.

[120] J. Muller-Brockmann, *Grid systems in graphic design: A visual communication manual for graphic designers, typographers and three dimensional designers*. Arthur Niggli, 1985.

[121] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[122] R. Goldenberg, *Twenty Years of the NBA Redrafted*, https://pudding.cool/2017/03/redraft/, 2017.

[123] M. Maclean, *D3.js multi-line graph with automatic (interactive) legend*, https://bl.ocks.org/d3noob/08af723fe615c08f9536f656b55755b4, Jul. 2014.

[124] R. Yeip, S. A. Thompson, and W. Welch, *A Field Guide to Red and Blue America*, http://graphics.wsj.com/elections/2016/field-guide-red-blue-america/, Jul. 2016.

[125] *Adobe Photoshop*, http://www.adobe.com/products/photoshop.html, 2020.

[126] *Adobe InDesign*, http://www.adobe.com/products/indesign.html, 2020.

[127] Figma Design, *Figma: The collaborative interface design tool*, https://www.figma.com/, 2020.

[128] Adobe Systems Inc., *How to blend objects in illustrator*, https://helpx.adobe.com/illustrator/using/blending-objects.html, Feb. 2017.

[129] Anima App., *Auto-layout: Introducing stacks-flexbox for sketch*, https://medium.com/sketch-app-sources/auto-layout-introducing-stacks-flexbox-for-sketch-c8a11422c7b5, Feb. 2017.

[130] Adobe Systems Inc., *Draw circular (polar) grids*, https://helpx.adobe.com/illustrator/using/drawing-simple-lines-shapes.html, Sep. 2017.

[131] W. Javed and N. Elmqvist, "Exploring the design space of composite visualization," in *Pacific Visualization Symposium, IEEE*, ser. PacificVis '12, IEEE, 2012, pp. 1–8. DOI: 10.1109/PacificVis.2012.6183556.

[132] J. Heer, D. Moritz, M. Correll, and K. Wongsuphasawat, *Vega Datalib*, https://github.com/vega/datalib, Oct. 2017.

[133] J. H. T. Claessen and J. J. van Wijk, "Flexible linked axes for multivariate data visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2310–2316, Dec. 2011. DOI: 10.1109/TVCG.2011.201.

[134] J. Lehni and J. Puckey, *Paper.js*, http://paperjs.org/, Oct. 2015.

[135] ReactiveX, *RxJS*, http://reactivex.io/rxjs/, Dec. 2016.

[136] DocumentCloud, *Backbone.js*, http://backbonejs.org/, Apr. 2016.

[137] New York Times, *2018: The Year in Visual Stories and Graphics*, https://www.nytimes.com/interactive/2018/us/2018-year-in-graphics.html, 2018.

[138] The Pudding, *The Pudding Archives*, https://pudding.cool/archives/, 2019.

[139] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko, "Effectiveness of animation in trend visualization," *IEEE transactions on visualization and computer graphics*, vol. 14, no. 6, pp. 1325–1332, 2008.

[140] M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe, "A comparative evaluation of animation and small multiples for trend visualization on mobile phones," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 364–374, 2019.

[141] D. Archambault, H. Purchase, and B. Pinaud, "Animation, small multiples, and the effect of mental map preservation in dynamic graphs," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 4, pp. 539–552, 2010.

[142] D. Guilmaine, C. Viau, and M. J. McGuffin, "Hierarchically animated transitions in visualizations of tree structures," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 2012, pp. 514–521.

[143] N. Bremer, *Visual Cinnamon*, http://www.visualcinnamon.com/, 2020.

[144] N. Halloran, *Neil Halloran*, http://www.neilhalloran.com/, 2020.

[145] S. Wu, *Shirley Xueyang Wu*, http://sxywu.com/, 2020.

[146] M. Stefaner, *Truth & Beauty*, http://truth-and-beauty.net/, 2020.

[147] J. Schwabish, *4 Observations on Animating Your Data Visualizations*, https://link.medium.com/OBs05OgeD3, 2019.

[148] L. Groeger, *That's the Power of Loops*, http://www.youtube.com/watch?v=zd0YQAgu3dI, 2015.

[149] A. Thomas, *The Timing of Baby Making*, https://pudding.cool/2017/05/births/, May 2017.

[150] G. Gianordoli, L. Salaberry, A. Biernath, and U. Syam, *I'm Not Feeling Well*, http://www.imnotfeelingwell.com/, 2019.

[151] S. Yee and T. Chu, *A visual introduction to machine learning*, http://www.r2d3.us/visual-intro-to-machine-learning-part-1/, Jul. 2015.

[152] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960. DOI: 10.1177/001316446002000104.

[153] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977. DOI: 10.2307/2529310.

[154] N. Bremer and M. Ranzijn, *Urbanization in East Asia between 2000 and 2010*, http://nbremer.github.io/urbanization/, Mar. 2015.

[155] E. L. Hutchins, J. D. Hollan, and D. A. Norman, "Direct manipulation interfaces," *Human-computer interaction*, vol. 1, no. 4, pp. 311–338, 1985. DOI: 10.1207/s15327051hci0104_2.

[156] D. Norman, *The Design of Everyday Things: Revised and Expanded Edition*. Basic books, 2013.

[157] A. Satyanarayan, B. Lee, D. Ren, J. Heer, J. Stasko, J. Thompson, M. Brehmer, and Z. Liu, "Critical Reflections on Visualization Authoring Systems," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 461–471, 2020. DOI: 10.1109/TVCG.2019.2934281.

[158] J. Harper and M. Agrawala, "Converting Basic D3 charts into Reusable Style Templates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 3, pp. 1274–1286, 2017. DOI: 10.1109/TVCG.2017.2659744.

[159] M. Guzdial, "Education paving the way for computational thinking," *Communications of the ACM*, vol. 51, no. 8, pp. 25–27, 2008.

[160] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2011, pp. 3363–3372. DOI: 10.1145/1978942.1979444.

[161] T. Munzner, *Visualization Analysis and Design*. AK Peters/CRC Press, 2014.

[162] J. Hoffswell, W. Li, and Z. Liu, "Techniques for flexible responsive visualization design," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)s*, ser. CHI '20, Honolulu, HI, USA: Association for Computing Machinery, 2020, pp. 1–13. DOI: 10.1145/3313831.3376777.

[163] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko, *Data Illustrator: Gallery*, http://data-illustrator.com/gallery.php, 2018.

[164] J. Thompson, Z. Liu, and J. Stasko, *Data Illustrator: Gallery*, http://data-animator.com/gallery/index.html, 2020.

[165] E. Hoque and M. Agrawala, "Searching the visual style and structure of d3 visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 1236–1245, 2020. DOI: 10.1109/TVCG.2019.2934431.

[166] J. Harper and M. Agrawala, "Deconstructing and Restyling D3 Visualizations," in *Proceedings of the ACM Symposium on User Interface Software and Technology*, ser. UIST '14, 2014, pp. 253–262. DOI: 10.1145/2642918.2647411.

[167] A. Srinivasan, S. M. Drucker, A. Endert, and J. Stasko, "Augmenting visualizations with interactive data facts to facilitate interpretation and communication," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 672–681, 2019. DOI: 10.1109/TVCG.2018.2865145.

[168] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," in *Proceedings of the IEEE Symposium on Visual Languages*, 1996, pp. 336–343. DOI: 10.1109/VL.1996.545307.

[169] J. S. Yi, Y. a. Kang, J. Stasko, and J. A. Jacko, "Toward a deeper understanding of the role of interaction in information visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1224–1231, 2007. DOI: 10.1109/TVCG.2007.70515.

[170] F. Hohman, M. Conlen, J. Heer, and D. H. (Chau, "Communicating with interactive articles," *Distill*, 2020, https://distill.pub/2020/communicating-with-interactive-articles. DOI: 10.23915/distill.00028.

[171] J. Zong, D. Barnwal, R. Neogy, and A. Satyanarayan, "Lyra 2: Designing interactive visualizations by demonstration," *To appear in IEEE Transactions on Visualization and Computer Graphics*, 2020.

[172] Aisch, Gregor and Cox, Amanda and Quealy, Kevin, *You Draw It: How Family Income Predicts Children's College Chances*, https://www.nytimes.com/interactive/2015/05/28/upshot/you-draw-it-how-family-income-affects-childrens-college-chances.html, 2015.

[173] Katz, Josh, *You Draw It: Just How Bad Is the Drug Overdose Epidemic?* https://www.nytimes.com/interactive/2017/04/14/upshot/drug-overdose-epidemic-you-draw-it.html, 2017.

[174] Buchanan, Larry and Park, Haeyoun and Pearce, Adam, *You Draw It: What Got Better or Worse During Obama's Presidency*, https://www.nytimes.com/interactive/2017/01/15/us/politics/you-draw-obama-legacy.html, 2017.

[175] Y.-S. Kim, K. Reinecke, and J. Hullman, "Explaining the gap: Visualizing one's predictions improves recall and comprehension of data," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '17, 2017, pp. 1375–1386.

[176] Midwest Uncertainty Collective and Northwestern University Knight Lab, *They Draw It!* http://mucollective.co/theydrawit/, 2015.

[177] E. M. Peck, S. E. Ayuso, and O. El-Etr, "Data is Personal: Attitudes and Perceptions of Data Visualization in Rural Pennsylvania," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, ser. CHI '19, Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–12, ISBN: 9781450359702. DOI: 10.1145/3290605.3300474.

[178] S. Almukhtar, A. Aufrichtig, M. Bloch, and J. Calderone, *Covid in the U.S.: Latest Map and Case Count*, https://www.nytimes.com/interactive/2020/us/coronavirus-us-cases.html, 2020.

[179] A. Bhatia, M. Conlen, F. Hohman, and S. Stalla, *Your Personal Carbon History*, https://parametric.press/issue-02/carbon-history/, 2020.